MI COMPUTER

35

CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Editorial Delta, S.A.

150 ptas.

MI COMPUTER CURSO PRACTICO
DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona, y comercializado en exclusiva por Distribuidora Olimpia, S.A., Barcelona

Volumen III - Fascículo 35

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones, Francisco Martín
Asesor técnico: Jesús Nebra

Redactores y colaboradores: G. Jefferson, R. Ford, S. Tarditti, A. Cuevas, F. Blasco

Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Paseo de Gracia, 88, 5.°, Barcelona-8
Tels. (93) 215 10 32 / (93) 215 10 50 - Télex 97848 EDLTE

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador,* se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-85822-94-3 (tomo 3)
84-85822-82-X (obra completa)
Depósito Legal: B. 52/1984

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 128409
Impreso en España - Printed in Spain - Septiembre 1984

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, Madrid-34.
Distribuye para Argentina: Viscontea Distribuidora, S.C.A., La Rioja 1134/56, Buenos Aires.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio Blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Ferrenquín a Cruz de Candelaria, 178, Caracas, y todas sus sucursales en el interior del país.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 16 690 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 087 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 3371872 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Distribuidora Olimpia (Paseo de Gracia, 88, 5.°, Barcelona-8), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Distribuidora Olimpia, en la forma establecida en el apartado b).

Para cualquier aclaración, telefonear al (93) 215 75 21.

No se efectúan envíos contra reembolso.





El poder de la luz

Cortesía de Teletape Video Ltd

Ian McKinnell

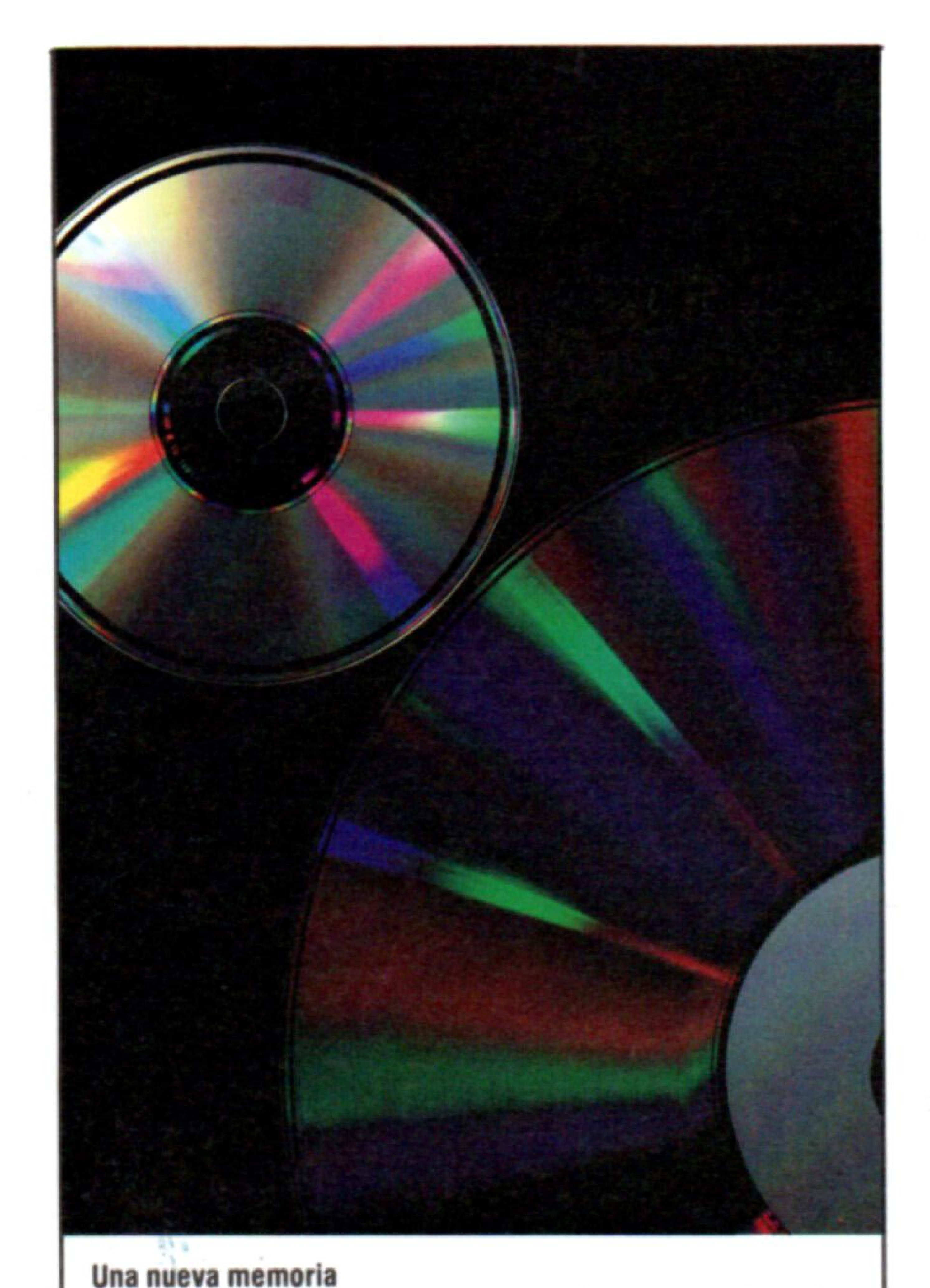
Combinación de vanguardia
Los aparatos reproductores de discos láser pronto serán tan asequibles como los ordenadores personales y será habitual encontrar a ambos conectados en el hogar. Comprando los discos y el software apropiados, el usuario podrá tener acceso a enormes bases de datos ilustradas y a sofisticados programas de entrenamiento, así como a entretenidos juegos de aventuras

Los discos láser son revolucionarios dispositivos a los que el usuario de microordenadores, por el descenso de los precios, ya puede tener acceso

A muchas personas les sorprenderá saber que los aparatos reproductores de discos láser, que hasta hace unos años se consideraban artículos de lujo por su elevado precio, ahora cuestan menos que los aparatos de video y ofrecen una reproducción visual muy superior. Una imagen de televisión es una imagen dinámica que se puede grabar en una cinta de video como una secuencia ininterrumpida. La única manera de localizar una determinada parte de la secuencia consiste en bobinar la cinta hacia adelante, ya sea a velocidad *play* (reproducción) o bien utilizando *fast forward* (avance rápido), con el contador de cinta como guía. Los discos almacenan las imágenes como fotogramas separados, lo que le permite al usuario acceder directamente, con precisión y rapidez, a cualquiera de ellos. La posición en disco de un fotograma se puede describir en términos de pista y sector, y un microprocesador puede llevar un catálogo actualizado de las posiciones.

Las tareas del microprocesador de activar una platina portadiscos a velocidades constantes y elevadas y posicionar una cabeza de reproducción exactamente sobre ciertas posiciones de la superficie, no son los principales logros de la tecnología del disco. El mayor desafío fue el de facilitar la inmensa densidad de almacenamiento de los discos. Posibilitar el almacenamiento de millares de fotogramas en un disco del tamaño de un disco fonográfico de larga duración de 12″ constituyó una labor especialmente audaz y ardua.

Si usted ha utilizado alguna vez gráficos de alta resolución en su micro, entonces sabrá que una imagen de televisión se compone de puntos individuales (pixels) de luz (cuanto mayor sea el número de puntos en la pantalla, mejor será la imagen) y que el almacenamiento de visualizaciones en pantalla de alta resolución exige emplear muchísima memoria. El BBC Micro, por ejemplo, tiene una reso-



Una nueva memoria
Los discos láser de doce pulgadas y los discos compactos más pequeños se pueden utilizar para almacenar información tanto de video como de audio, así como los datos digitales almacenados en cassettes y discos flexibles de ordenador. El inconveniente es que en los discos no se puede escribir y que técnicamente son sólo ROM. Esto significa que el usuario no puede guardar sus propios programas e información en un disco láser.

Ian McKinnell

lución máxima de 640 × 256 pixels por visualización en pantalla, que ocupa exactamente 20 Kbytes de memoria. Si un fotograma de televisión tuviera una resolución no más refinada que ésta, entonces almacenar un segundo de video (a 25 fotogramas por segundo) ¡requeriría 25 × 20 Kbytes de almacenamiento! Un minuto de programa de televisión ocuparía 30 Megabytes (30 millones de bytes) y un episodio de una serie de televisión necesitaría más de un Gigabyte (1 000 Megabytes) de memoria; y si decidiéramos emplear discos flexibles de densidad simple y de una sola cara, que utilizan la mayoría de los ordenadores personales, ¡exigiría más de 6 000 discos y una semana de trabajo!

A la luz de estas cifras, el almacenamiento en cinta de video se vuelve bastante más atractivo: trasladar media hora de programas de televisión a un disco parece plantear problemas insuperables.

La respuesta a este dilema radica en escribir los datos muy pequeños. Una grabadora de discos láser graba los datos con un haz láser delgado como un cabello sobre una placa metálica revestida con una envoltura translúcida resistente. Para leer los datos del disco se utiliza un haz de bajo poder. Para este disco se emplean láseres como aguja de lectura-escritura por ser dispositivos de fina resolución y baja tolerancia. Ninguna otra técnica podría leer y escribir tal cantidad de datos en un espacio tan pequeño.

El formato es una combinación de las técnicas utilizadas en alta fidelidad y en las unidades de disco. Los surcos de los discos fonográficos forman una espiral continua y las paredes de los surcos llevan una representación grabada de las formas de onda del sonido grabado. La mayoría de los usuarios de micros saben que las pistas de una unidad de disco son círculos concéntricos, y que en el disco la información se escribe magnéticamente y se almacena digitalmente como patrones de unos y ceros. El formato del disco láser utiliza pistas y no surcos, pero éstas forman una espiral. En el disco la información se graba ópticamente en patrones de unos y ceros, pero no se puede borrar. Los unos y los ceros (que representan los patrones de puntos de que consta la imagen de televisión) se escriben en la superficie del disco mediante el láser, que quema diminutos orificios en la película metálica para representar a los unos, dejándola intacta para representar los ceros. Cada agujero tiene una anchura de medio micrómetro (0,0005 mm) y una profundidad de un décimo de micrómetro (0,0001 mm). Así, un centímetro cuadrado de superficie de disco podría albergar 400 millones de estos orificios.

Esta increíble miniaturización apenas si es suficiente para hacer frente a las demandas de almacenamiento de video. Un disco de 35 cm de diámetro retiene 54 000 fotogramas de televisión por cada cara, o aproximadamente 36 minutos de tiempo de reproducción. Los cálculos del tamaño de los fotogramas que realizamos antes en este artículo se basaban en la resolución de gráficos en ordenador en blanco y negro, mientras que el disco láser ha de almacenar información en color para cada punto de la imagen de televisión y llevar, asimismo, un canal de audio. Un fotograma en color, con su correspondiente canal de sonido, podría, por consiguiente, requerir 100 Kbytes de espacio de almacenamiento. 54 000 de estos fotogramas utilizarían hasta 5 400 000 Kbytes, o 5,4 Gbytes.

Habiendo superado el problema de la limitación de almacenamiento, los discos láser ofrecen amplias posibilidades de aplicaciones. Uno de los beneficios fundamentales es el de eliminar uno de los mayores obstáculos del procesamiento de datos: la recogida y entrada de datos. La información por lo general está disponible fácilmente, pero todavía alguien debe sentarse frente a un terminal y digitar representaciones codificadas de los datos en el sistema: un procedimiento tedioso, caro y que ocupa mucho tiempo. Si, en cambio, uno puede dirigir una cámara a los datos y dejar que la misma almacene la información visual en disco mientras uno sólo da entrada a detalles de indexación de las imágenes grabadas, entonces el volumen de trabajo disminuye sustancialmente.

Los aparatos reproductores de discos láser varían mucho en cuanto a su sofisticación. Se encuentran a la venta aparatos económicos para uso personal. Éstos se pueden utilizar para visionar películas como un aparato de video normal, pero con la ventaja adicional de que proporcionan una notable calidad de imagen, tanto en la reproducción normal como en congelación de imagen y cámara lenta.

Las verdaderas posibilidades no empiezan hasta que uno puede hacer que los fotogramas individuales los seleccione un programa para ordenador. Los aparatos reproductores que pueden hacer esto son de marcas como Pioneer y Philips, pero por el momento son caras y están destinadas sólo a la utilización profesional. El sistema más sencillo consiste en usar una interface IEEE o RS232 de modo que



el ordenador pueda seleccionar un fotograma determinado por su número.

Philips ha llevado la idea un paso más adelante y ha incorporado en sus modelos más recientes un microordenador sencillo. Éste puede cargar un programa apropiado para un disco determinado desde un cartucho EPROM enchufado en la máquina, o bien desde el propio disco láser. Cada disco láser almacena dos canales de audio y una pista de video. Esto permite, por ejemplo, que un mismo disco contenga una pista de sonido en dos idiomas. Sin embargo, si no se necesita la segunda pista de audio, se la puede utilizar para almacenar un programa de ordenador.

De modo que contamos con un banco de 54 000 imágenes de calidad bajo el control del ordenador. La etapa final consiste en mezclar las imágenes del disco láser con el texto proveniente del ordenador. Esto se podría hacer con dos monitores separados, o mezclando dos entradas de video, o utilizando un monitor con su propio generador de teletexto. Esta fase final constituye un medio totalmente nuevo: el video interactivo. El usuario y el software pueden guiar la visualización en televisión, tanto de secuencias de acción como de fotogramas fijos, mediante la lectura del disco por cualquier orden.

La aplicación más obvia es para una base de datos ilustrada. El usuario podría formular preguntas al ordenador y éste recuperaría la información pertinente de una base de datos e instruiría al aparato reproductor para que visualizara un fotograma de video adecuado. Se podría utilizar como referencia en bibliotecas y en escuelas, para todo tipo de fines, desde identificar diversas flores hasta seleccionar artículos de un catálogo.

El video interactivo puede progresar aún hasta otra etapa al implicar al usuario en las secuencias mostradas en la pantalla. Un programa de entrenamiento podría explicar algo en un breve trozo de película, recapitulando de ser necesario o entrando en más detalles, y así sucesivamente. Se podría incluso producir películas en las que el usuario asumiera la tarea de dirección tomando sus propias decisiones en nombre de los personajes de la historia. El desarrollo de la película, como su final, serían distintos según cómo uno la "jugara". Los entusiastas de los juegos de aventuras para micros personales encontrarán apasionantes estas posibilidades.

En el desarrollo de este mercado existen, por supuesto, ciertos problemas. Aparte del costo de los equipos y de la fabricación de discos láser, se han de desarrollar nuevas capacidades en cuanto al diseño y la producción de discos interactivos, tanto desde el punto de vista del software para ordenadores como del guión y filmación de imágenes. Muchas empresas pequeñas están empezando a afrontar estos desafíos; es así como una multinacional denominada Computer Assisted Televideo (CAT) ya se ha establecido como una fuerza dominante en el negocio. La empresa le proporciona al cliente un servicio completo: elección e instalación de equipos, diseño y producción de discos y escritura del software correspondiente. CAT tiene planificado abordar casi todas las aplicaciones del video interactivo, pero los elevados costos actuales limitan la mayor parte de su trabajo a la producción de programas de entrenamiento para grandes empresas.

No obstante, las posibilidades para el usuario de un micro personal no se deben ignorar. A pesar de que el costo de producción de los discos es elevado, no lo es más que los costos relacionados con películas y grabadoras. Los discos láser ya están a la venta, de modo que los discos interactivos, con software, podrán adquirirse pronto, aunque, eso sí, a un precio bastante mayor. Considerando la calidad de los programas de entretenimiento y educativos que posibilita el video interactivo, el precio a pagar puede considerarse razonable.



El video interactivo
Muchos aparatos reproductores de discos láser poseen una interface IEEE o RS232 que les permite ser controlados por microordenadores. Una instalación típica podría ser un ordenador con una base de datos relacionada con imágenes del disco láser, almacenadas en discos flexibles. El usuario puede seleccionar ítems de interés de la base de datos, el ordenador instruye luego al aparato reproductor de discos láser para que busque y visualice la imagen correspondiente en virtud de un número de fotograma. El sistema aquí ilustrado adopta un enfoque popular al combinar la salida del ordenador y las imágenes del disco láser en una sola pantalla. Un sistema más sencillo utilizaría dos pantallas separadas

Steve Cross

Desfile ordenado

Los archivos secuenciales son herencia de un tiempo en que el procesamiento de datos se basaba en cinta. Veamos cómo se crean y se accede a ellos desde un programa en BASIC

El archivo binario (o de programas) es un caso simplificado de archivo secuencial. Cuando usted digita SAVE "nombreprog", el sistema operativo realiza las diversas operaciones necesarias en la creación de un archivo: primero, abre comunicación con la cinta o la unidad de disco y escribe una etiqueta de encabezamiento que contiene el nombre del archivo y algo de información acerca de su contenido. A continuación, el sistema operativo escribe en el archivo el bloque de memoria que contiene el programa corriente. Por último, escribe en el archivo un indicador de final de archivo y cierra la comunicación entre el ordenador y la cinta o unidad de disco.

Las órdenes de BASIC que se utilizan para crear un archivo secuencial varían de un ordenador a otro, pero todas ellas deben llevar a cabo las mismas tareas. Tomando como ejemplo al Commodore 64:

```
100 R$ = "Éste es un registro del archivo"
150 OPEN 5,1,2,"ARCHDATO,SEQ,WRITE"
200 PRINT #5,R$
250 CLOSE 5
```

Se empieza con la orden OPEN 5,1,2 para que el sistema operativo establezca un canal de comunicación, denominado canal 5, con el dispositivo número 1, la unidad de cinta. El sistema operativo puede crear archivos secuenciales en un número de dispositivos distintos, y puede acceder a varios archivos diferentes al mismo tiempo, de modo que el canal del ordenador al dispositivo y, por tanto, a un archivo determinado, debe estar etiquetado. (En cuanto al último número, 2, es específico del Commodore y no importa por ahora.)

Después de la orden OPEN viene el nombre del archivo. En el Commodore, éste consiste en un nombre como ARCHDATO" seguido por el tipo de archivo (SEQ, por archivo secuencial) y el método de acceso (WRITE, que indica que el archivo se abre para escribir). Los archivos secuenciales, en efecto, se pueden abrir (OPEN) ya sea para escribir, ya sea para leer, y no para ambas cosas al mismo tiempo. Esta orden activa la cabeza de lectura-escritura del dispositivo, y escribe un registro de "encabezamiento" compuesto por el nombre del archivo y alguna información del sistema para indicar el comienzo del archivo.

La orden PRINT #5 de la línea 200 envía datos al archivo. PRINT mantiene su significado habitual, pero el signo "#" indica que se han de enviar datos al canal especificado y no a la pantalla, que es el canal de salida en caso de que no haya indicación alguna. El contenido de R$, por consiguiente, se envía por el canal 5 al archivo secuencial denominado "ARCHDATO". Ahora el archivo contiene un registro: la serie "Éste es un registro de archivo". Por último, CLOSE 5 cierra el canal 5 para toda otra comunicación más, y escribe en el archivo un indicador de final de archivo.

La información de un archivo puede leerse en una segunda etapa de este modo:

```
500 OPEN 3,1,2,"ARCHDATO,SEQ,READ"
550 INPUT #3,A$
600 CLOSE 3
650 PRINT A$
```

Tanto este fragmento de programa como el anterior utilizan la orden OPEN, que aquí significa "hallar el comienzo del archivo denominado ARCHDATO y prepararse para leerlo", mientras que previamente significaba "crear un archivo denominado ARCHDATO y prepararse para escribir en él". Los números de canal son distintos, pero éstos simplemente son etiquetas (en ambos casos se podría haber escogido un número diferente sin que se afectara la operación); la dirección del dispositivo, no obstante, es la misma en ambos casos porque el archivo está almacenado en el dispositivo 1, la unidad de cinta. El nombre de archivo y el tipo de archivo son los mismos porque identifican el archivo a acceder, pero el método de acceso es diferente: READ en lugar de WRITE.

Este programa tiene INPUT #3,A$, que significa "entrada desde canal 3" en vez de desde el teclado, el canal de entrada implícito cuando no se especifica otro. El primer registro completo del archivo se lee y se envía desde el archivo a través del canal 3 a la variable A$, de la misma manera como se envió, a través del canal 5 desde la variable R$ el archivo mediante PRINT #5,R$.

De este modo se evidencian algunas de las limitaciones y la mayoría de las ventajas de los archivos secuenciales: pueden almacenar todo cuanto pueda retener una variable y no existen limitaciones en cuanto a las dimensiones del archivo o su estructura. Pero su contenido ha de leerse empezando por el principio del archivo, registro a registro; no hay ninguna forma de abrir el archivo por un registro determinado, ni saltar, releer, borrar, insertar ni añadir nada en él.



Dentro de un orden
Los archivos secuenciales se desarrollaron teniendo presente el almacenamiento en cinta; pero el orden en el que se envían los registros al archivo debe ser respetado. La única forma de clasificar o editar un archivo secuencial es mediante la creación de una nueva versión del mismo en algún otro lugar de la cinta, así como una cassette de música se puede editar sólo regrabándola o cortando la cinta

Bob Hall

Archivar y hallar

```
199 REM ****************** CMB C64 ******************
200 REM*                ESCRIBIR ARCHIVOS              *
201 REM ****************** CBM C64 ******************
220 GOSUB 1500
240 FOR K = 65 TO 90
260 Z$ = CHR$(K)+X$:C$ = D$+"ESCRIBIENDO   " + CHR$(K)
280 GOSUB 2000
300 NEXT K
399 REM ********************************************
400 REM*                 LEER ARCHIVOS                  *
401 REM ********************************************
420 FOR L = 0 TO 1 STEP 0:FOR M = 1 TO 1
440 PRINT D$;"ACCEDER A REGISTROS"
460 INPUT"BUSCAR SERIE (* = ABANDONAR   )";N$
480 L$ = LEFT$(N$,1):IF L$ = "*" THEN GOTO 5000
500 IF L$ < "A" OR L$ > "Z" THEN M = 0
520 NEXT M
540 Z$ = L$ + Y$
560 GOSUB 3000
580 PRINT TAB(5) "REGISTRO   ";N;"    (PULSAR CUALQUIER
    TECLA)"
600 GET GT$:IF GT$ = "" THEN 600
620 NEXT L
999 END
1499 REM ******************************************
1500 REM ************ S/R INICIALIZAR ************
1501 REM ******************************************
1520 D$ = CHR$(147):PRINT D$,CHR$(8);CHR$(142)
1540 X$ = ",S,W":Y$ = ",S,R"
1600 RETURN
1999 REM ******************************************
2000 REM ********* S/R ESCRIBIR UN ARCHIVO *********
2001 REM ******************************************
2020 PRINT C$:INPUT"CUANTOS REGISTROS";R
2040 OPEN 8,8,2,Z$
2060 IF R = 0 THEN PRINT #8,"*":CLOSE8:RETURN
2080 FOR I = 1 TO R
2100 PRINT C$:PRINT "REGISTRO #";I
2120 INPUT "TEXTO.....";R$
2140 PRINT #8,R$
2160 NEXT I
2180 PRINT #8,"*":CLOSE8
2499 RETURN
2999 REM ******************************************
3000 REM *********** S/R LEER UN ARCHIVO ***********
3001 REM ******************************************
3020 PRINT D$;"BUSCANDO   ";L$;" A ";N$
3040 OPEN 8,8,2,Z$
3060 FOR I = 1 TO 100000
3080 INPUT #8,R$
3100 PRINT R$
3120 IF R$ = "*" THEN N = 0:I = 100000
3140 IF R$ = N$ THEN N = I:I = 100000
3160 NEXT I:CLOSE8:N$ = ""
3180 RETURN
5000 REM ************* CERRAR PROGRAMA *************
5020 PRINT CHR$(9);"FIN DEL PROGRAMA":STOP
```

220: Inicialización

260-280: Crea archivos de la "A" a la "Z"

420-540: Introduce registro de búsqueda, halla la letra inicial e identifica el archivo que la contiene

560: Busca ese archivo

580-600: Informa del resultado de la búsqueda

1520: Limpia la pantalla y establece la modalidad mayúsculas

2040: Abre (OPEN) un archivo para escribir (WRITE)

2060: Verifica un archivo vacío, escribe "*", cierra (CLOSE) el archivo

2120-2140: Introduce el texto del registro y lo escribe en el archivo

2180: Escribe "*" señal del último registro, y cierra (CLOSE) el archivo

3040: Abre (OPEN) el archivo para leer (READ)

3120: Comprueba el último registro y abandona la búsqueda

3140: Comprueba si se ha hallado el registro y abandona la búsqueda

3160: Cierra (CLOSE) el archivo

Este programa demuestra la utilización de los archivos secuenciales en disco mediante la creación de un libro con índice alfabético sencillo que consta de 26 archivos, uno para cada letra del alfabeto. En cada archivo se pueden digitar registros que comiencen con esa letra, o ningún registro en absoluto. Luego se puede buscar cualquier registro. El archivo adecuado se buscará y se visualizará hasta que se encuentre el registro; de no encontrarse, se visualizará el mensaje "REGISTRO 0". El programa utiliza el DOS para ganar acceso directo al archivo correspondiente al registro de búsqueda; pero el archivo propiamente dicho se lee secuencialmente. Esto reduce el tiempo de búsqueda a una longitud razonable; si los archivos estuvieran en cinta, buscarlos sería excesivamente lento

Complementos al BASIC

BBC Micro

Tenga en cuenta las variaciones para el Spectrum de las líneas 260 y 540. Sustituya PRINT #8, INPUT #8 y CLOSE8 por PRINT #C8, INPUT #8 y CLOSE #8

```
600 GT$ = GET$
1520 *DISK
1530 MODE 7
1540 D$ = CHR$(12):PRINT D$"UTILIZAR
     MAYUSCULAS"
1550 PRINT"—PULSAR CUALQUIER
     TECLA—":GT$ = GET$
2040 C8 = OPENOUT(Z$)
3040 C8 = OPENIN(Z$)
5020 PRINT"FIN DEL PROGRAMA"
```

Spectrum Microdrive

Ponga LET donde sea necesario.
Sustituya PRINT D$;.. por CLS PRINT..
Sustituya PRINT C$ por CLS:PRINT:C$
Sustituya OPEN 8,8,2Z$ por OPEN #8;"m";1;Z$
Sustituya CLOSE8 por CLOSE #8
Borre la línea 1540

```
260 Z$ = CHR$(K):LET C$ = "ESCRIBIENDO" + Z$
480 LET L$ = N$(1):IF L$ = "*" THEN GOTO
    5000
540 LET Z$ = L$
600 PAUSE 0
1520 CLS:LET F2 = PEEK 23658:POKE 23658,8
3080 INPUT #8;R$
5020 POKE 23658,F2:PRINT "FIN DEL
     PROGRAMA":STOP
```

Roy Ingrams

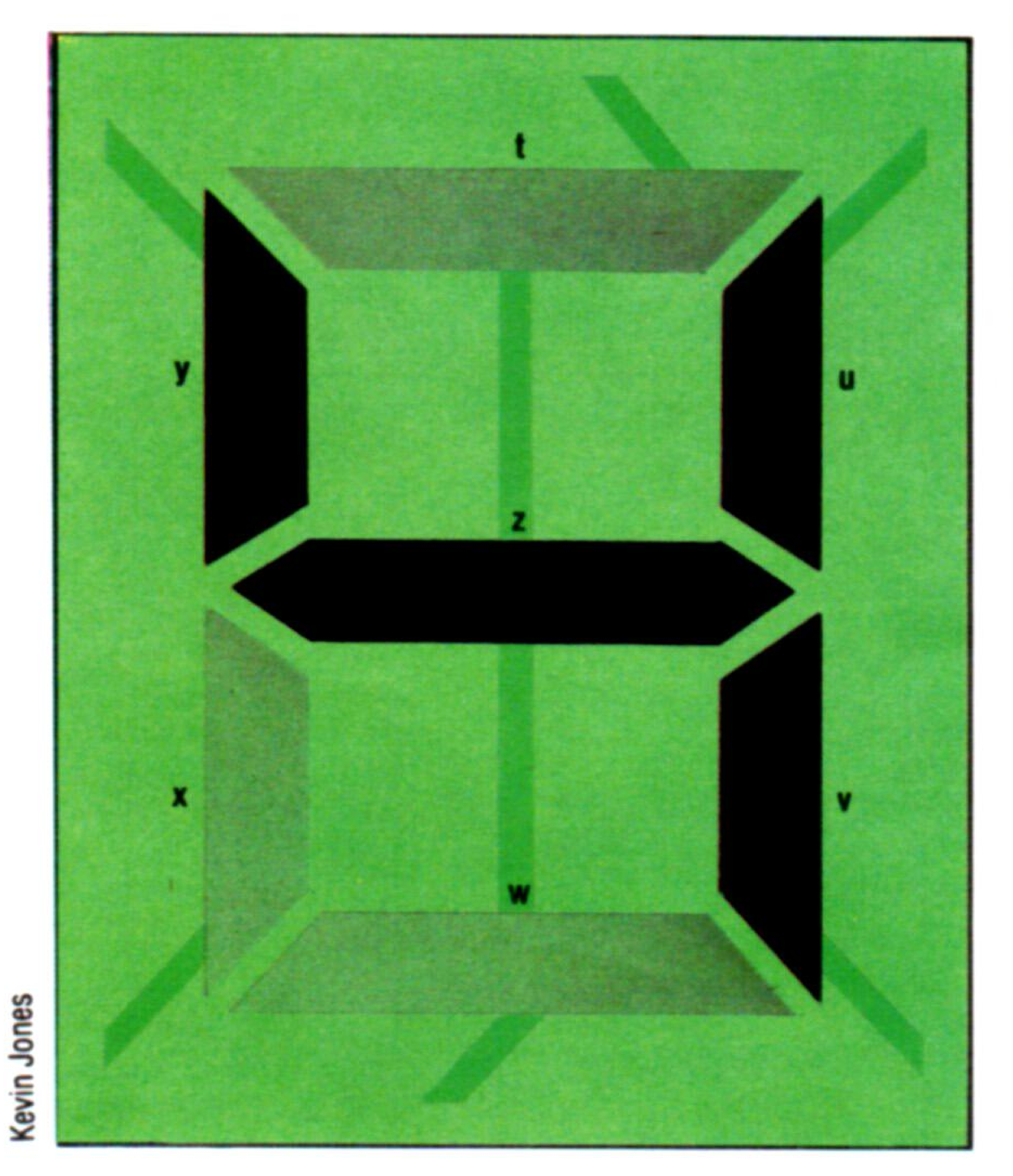
Éste es su número

Con siete luminosos segmentos pueden calculadoras, relojes y ordenadores portátiles representar cualquier número decimal

En nuestro diseño de circuito daremos por sentado que las representaciones binarias de los números decimales están en un código conocido como decimal codificado en binario, o BCD *(Binary Coded Decimal)*. Consiste en que cada dígito decimal tiene su equivalente en un grupo de cuatro dígitos binarios, tal como vemos en la tabla.

Una visualización en siete segmentos representa a los números decimales iluminando ciertos diodos emisores de luz, en el caso de una visualización LED (*Light-Emitting Diode*: diodo emisor de luz), o cambiando la polaridad de ciertas barras, como en las visualizaciones en cristal líquido (LCD: *Liquid Crystal Display*). La figura ilustra el nombre que hemos dado a cada uno de los siete segmentos (al segmento superior lo llamaremos *t*, al segundo inferior *w*, etc.). También indicamos qué segmentos se usan cada vez para formar los dígitos del 0 al 9.

Decimal codificado en binario	Decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010 a 1111	no se utilizan



La pantalla LCD comprende siete segmentos conmutables individualmente. Los designamos con letras de la *t* a la *z*

Kevin Jones

Ésta es toda la información que necesitamos para construir una tabla de verdad para nuestro conversor. A partir de la ilustración de la página contigua podemos decir qué segmentos se necesitan activar cuando el circuito recibe cada entrada individual. Por ejemplo, si la entrada binaria es 0100 (4 en decimal), el circuito activará los segmentos etiquetados *u, v, y* y *z*. Del mismo modo, la entrada 1000 (en notación decimal 8) debe hacer que se iluminen todos los segmentos. La tabla de verdad del circuito será:

Decimal	Entradas				Salidas						
	A	B	C	D	t	u	v	w	x	y	z
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
	1	0	1	0	X	X	X	X	X	X	X
	1	0	1	1	X	X	X	X	X	X	X
	1	1	0	0	X	X	X	X	X	X	X
	1	1	0	1	X	X	X	X	X	X	X
	1	1	1	0	X	X	X	X	X	X	X
	1	1	1	1	X	X	X	X	X	X	X

Por desgracia, no existe niguna forma sencilla de analizar esta tabla de verdad, y, por consiguiente, cada salida (*t, u, v, w, X, y* y *z*) se ha de simplificar por separado. Podemos construir un diagrama de Karnaugh para cada salida, colocando una X en cada diagrama para las zonas "indiferentes". Éstas podrían ayudar a la simplificación en algunos casos. Le hemos trazado aquí los siete diagramas k, uno para cada línea. Encerrando los grupos en óvalos podemos elaborar una expresión simplificada para cada línea de salida. A veces es posible incluso otra simplificación por medio del factor común.

El primer diagrama k, por ejemplo, trata de las entradas que activarán el segmento *t* (que se usa en todos los números salvo dos). Simplificadas las expresiones para todas las líneas de salida, se dibuja el circuito final.



0 1 2 3 4 5 6 7 8 9

Codificación de los segmentos
A partir de estos diagramas podemos determinar qué segmento es necesario activar para formar cada dígito. El dígito simple, "1", sólo necesita que se enciendan los segmentos *u* y *v*. La tabla de verdad de la página contigua nos proporciona las salidas necesarias para cada uno de los dígitos

Para *t*

	A	A	$\overline{A}$	$\overline{A}$	
B	X	X	1	1	C
$\overline{B}$	X	X	1	1	C
$\overline{B}$	1	1		1	$\overline{C}$
B	X	X	1		$\overline{C}$
	$\overline{D}$	D	D	$\overline{D}$	

$t=C+A+\overline{B}.\overline{D}+B.D$

Para *u*

	A	A	$\overline{A}$	$\overline{A}$	
B	X	X	1		C
$\overline{B}$	X	X	1	1	C
$\overline{B}$	1	1	1	1	$\overline{C}$
B	X	X		1	$\overline{C}$
	$\overline{D}$	D	D	$\overline{D}$	

$u=\overline{B}+C.D+\overline{C}.\overline{D}$

Para *y*

	A	A	$\overline{A}$	$\overline{A}$	
B	X	X	1	1	C
$\overline{B}$	X	X	1		C
$\overline{B}$	1	1	1	1	$\overline{C}$
B	X	X	1	1	$\overline{C}$
	$\overline{D}$	D	D	$\overline{D}$	

$v=D+\overline{C}+B$

Para *w*

	A	A	$\overline{A}$	$\overline{A}$	
B	X	X		1	C
$\overline{B}$	X	X	1	1	C
$\overline{B}$	1	1		1	$\overline{C}$
B	X	X	1		$\overline{C}$
	$\overline{D}$	D	D	$\overline{D}$	

$w=A+C.\overline{D}+\overline{B}.C+\overline{B}.\overline{D}+B.\overline{C}.D$

Para *x*

	A	A	$\overline{A}$	$\overline{A}$	
B	X	X		1	C
$\overline{B}$	X	X		1	C
$\overline{B}$	1			1	$\overline{C}$
B	X	X			$\overline{C}$
	$\overline{D}$	D	D	$\overline{D}$	

$x=C.\overline{D}+A.\overline{D}+\overline{B}.\overline{D}$
$=(A+\overline{B}+C).\overline{D}$

Para *y*

	A	A	$\overline{A}$	$\overline{A}$	
B	X	X		1	C
$\overline{B}$	X	X			C
$\overline{B}$	1	1		1	$\overline{C}$
B	X	X	1	1	$\overline{C}$
	$\overline{D}$	D	D	$\overline{D}$	

$y=A+B.\overline{C}+B.\overline{D}+\overline{C}.\overline{D}$
$=A+B.(\overline{C}+\overline{D})+\overline{C}.\overline{D}$

Para *z*

	A	A	$\overline{A}$	$\overline{A}$	
B	X	X		1	C
$\overline{B}$	X	X	1	1	C
$\overline{B}$	1	1			$\overline{C}$
B	X	X	1	1	$\overline{C}$
	$\overline{D}$	D	D	$\overline{D}$	

$z=A+\overline{B}.C+B.\overline{C}+B.\overline{D}$
$=A+\overline{B}.C+B.(\overline{C}+\overline{D})$

Liz Dixon

El circuito que hemos diseñado sirve tan sólo para una celda de la pantalla. Dado que la mayoría de los circuitos de este tipo poseen 8 o incluso 10 unidades, parecería que fuera necesario duplicar el circuito para cada celda. Sin embargo, se puede conseguir que todas las visualizaciones compartan un mismo conversor en virtud de un proceso que se conoce como *multiplexión*. Consiste en conmutar cada unidad de la visualización encendiéndola y apagándola en secuencia de modo que en cualquier momento dado sólo haya una unidad aceptando información proveniente del circuito del conversor. Debido a que esto sucede a una extremada velocidad, la visualización parece constante.

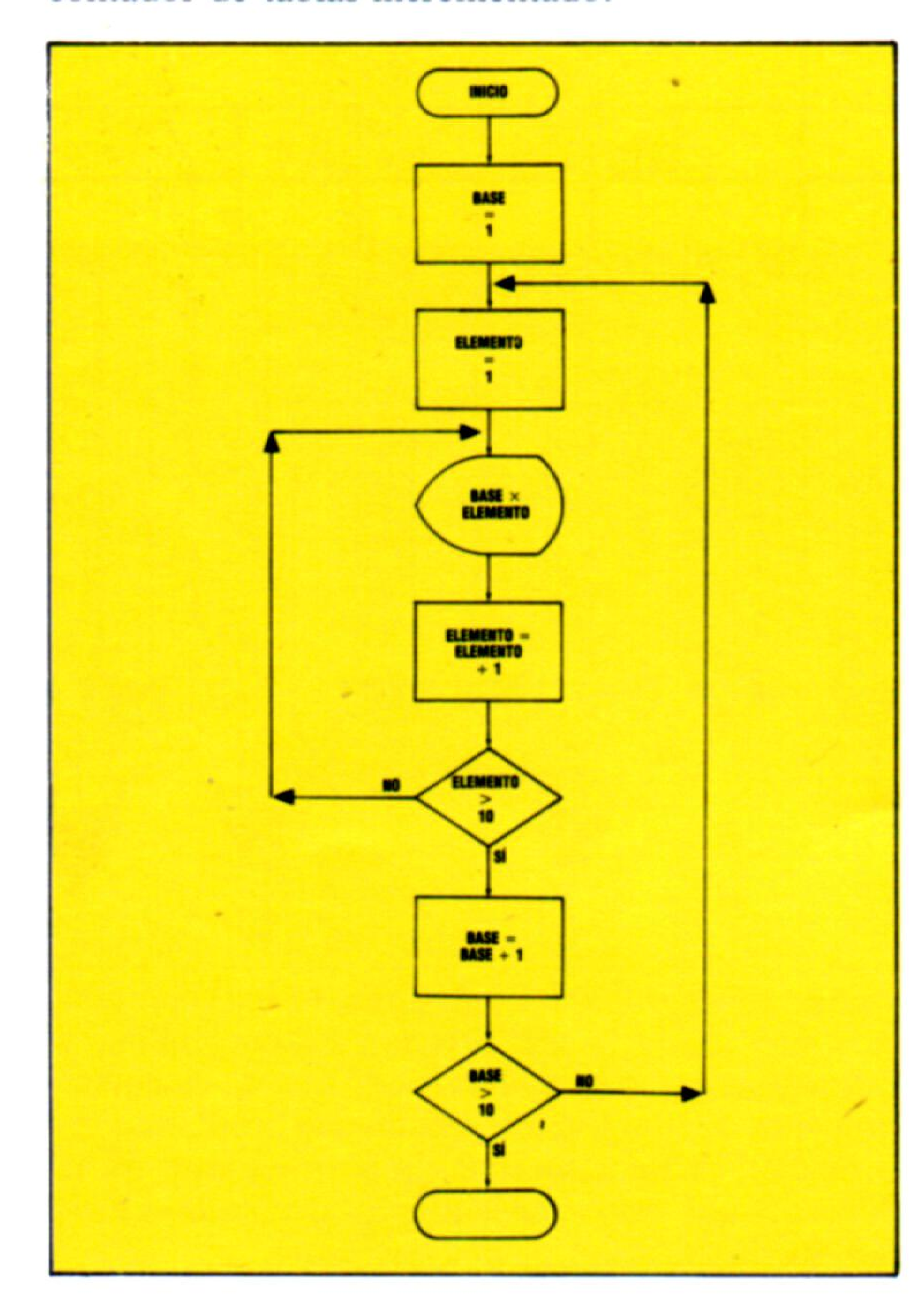
Contarlo todo

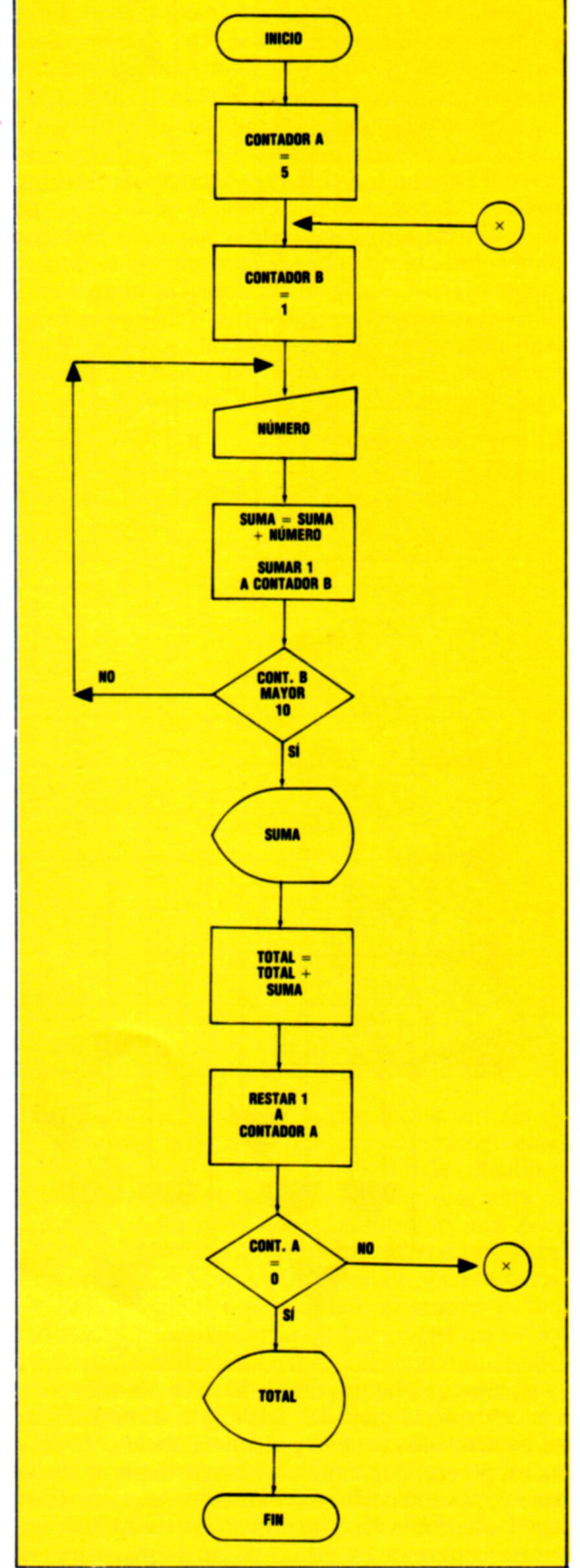
Completamos la lección iniciada en el capítulo anterior estudiando los contadores múltiples

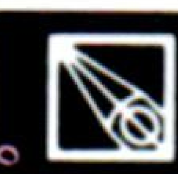
Al igual que en el caso de los bucles, un mismo programa puede contener un número indeterminado de contadores. Así, tomando nuevamente el ejemplo del capítulo anterior, efectuaremos algunas variaciones.

Se dispone ahora de cinco grupos de diez cantidades cada uno y, además de visualizarse las sumas parciales (por cada grupo), se debe dar, asimismo, la suma total de ellos. Para conseguirlo, esta vez va a ser necesario utilizar dos contadores: el A, que será el de los lotes de números y que reiterará cinco veces el ciclo general, y el B, que se encargará de repetir diez veces las operaciones necesarias para conseguir los totales parciales.

Para completar este tema le ofrecemos otro ejemplo de programa con más de un contador. Es una versión del conocido programa de la tabla de multiplicar al que se ha incluido alguna variación. Se trata, como siempre, de visualizar las tablas de los diez primeros elementos, del uno al diez; pero esta vez con una solución más elegante. Al igual que en el anterior ejemplo, van a emplearse dos contadores: el del número base, cuya tabla se va a visualizar, y el de elementos, que, a diferencia del anterior, cada vez que rebase su valor máximo de 10, volverá a ponerse con su valor inicial de uno, para ser utilizado con la próxima base, es decir, el contador de tablas incrementado.







Prueba

He aquí las respuestas a los ejercicios planteados en el capítulo anterior

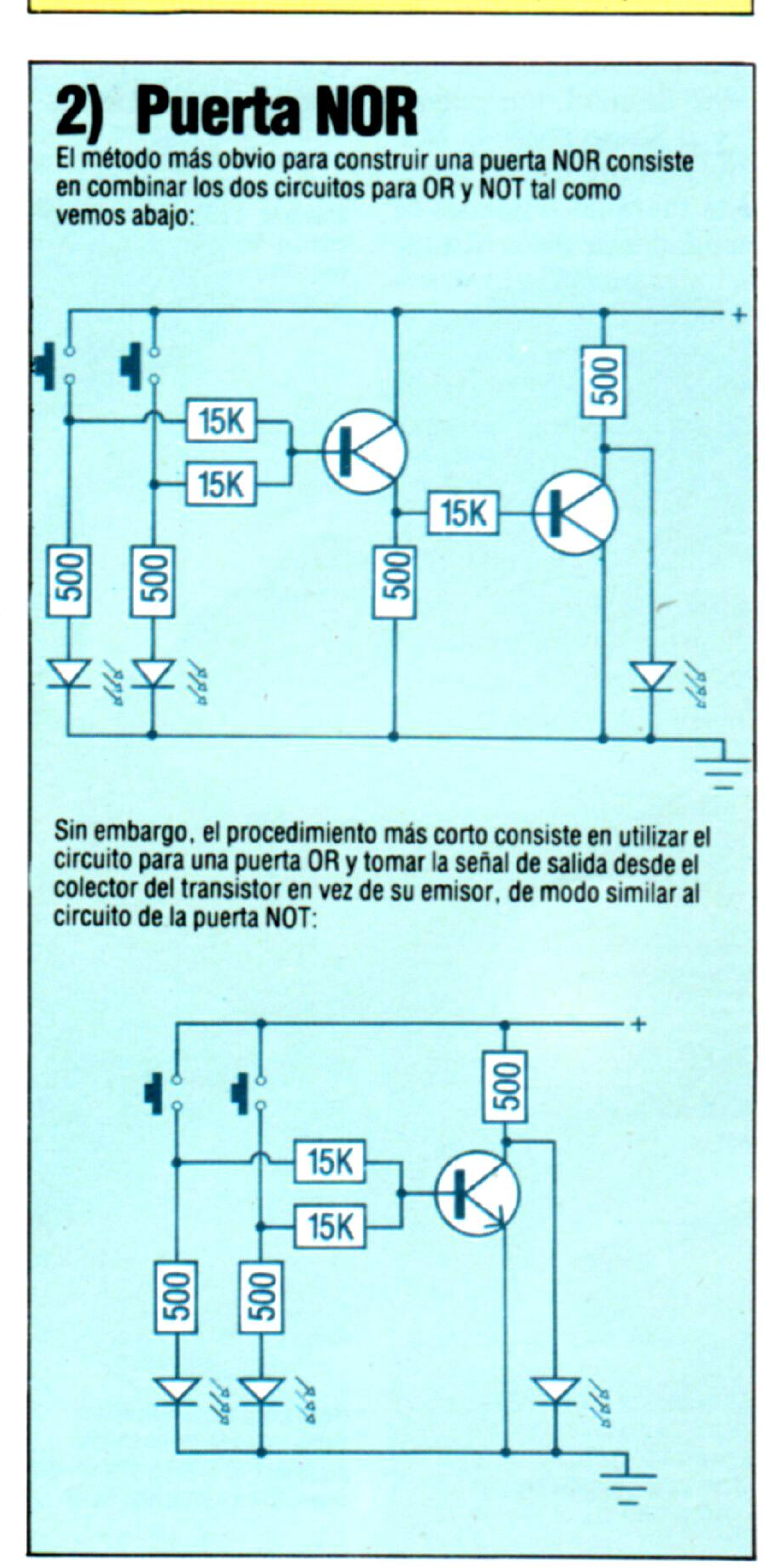
En el último capítulo de *Bricolaje* le ofrecimos algunos ejercicios sencillos para poner a prueba sus conocimientos de los temas que hemos tratado. Aquí le presentamos las respuestas. Puede que éstas no se correspondan exactamente con las suyas, ya que existen muchas maneras de diseñar un circuito para que realice una tarea determinada. No obstante, usted podrá probar si sus circuitos funcionan cotejándolos con las tablas de verdad dadas con el problema.

1) Resistencias

Los valores de las resistencias ilustradas son: a) 6 400 K-ohmios y b) 150 K-ohmios. Ésta tiene franjas marrón-verde-marrón (leyéndolas en dirección a una franja dorada o plateada).

2) Puerta NOR

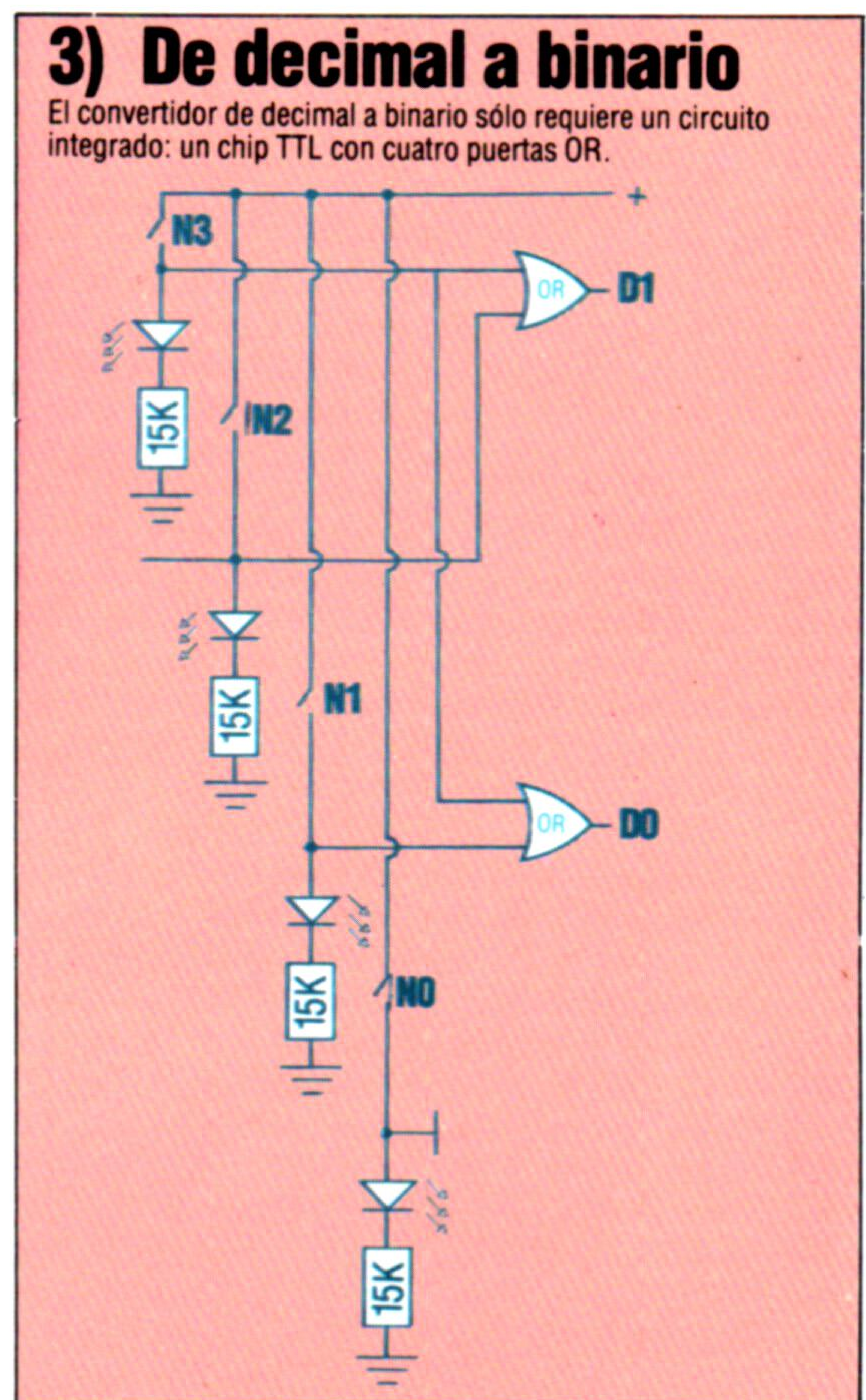
El método más obvio para construir una puerta NOR consiste en combinar los dos circuitos para OR y NOT tal como vemos abajo:



Sin embargo, el procedimiento más corto consiste en utilizar el circuito para una puerta OR y tomar la señal de salida desde el colector del transistor en vez de su emisor, de modo similar al circuito de la puerta NOT:

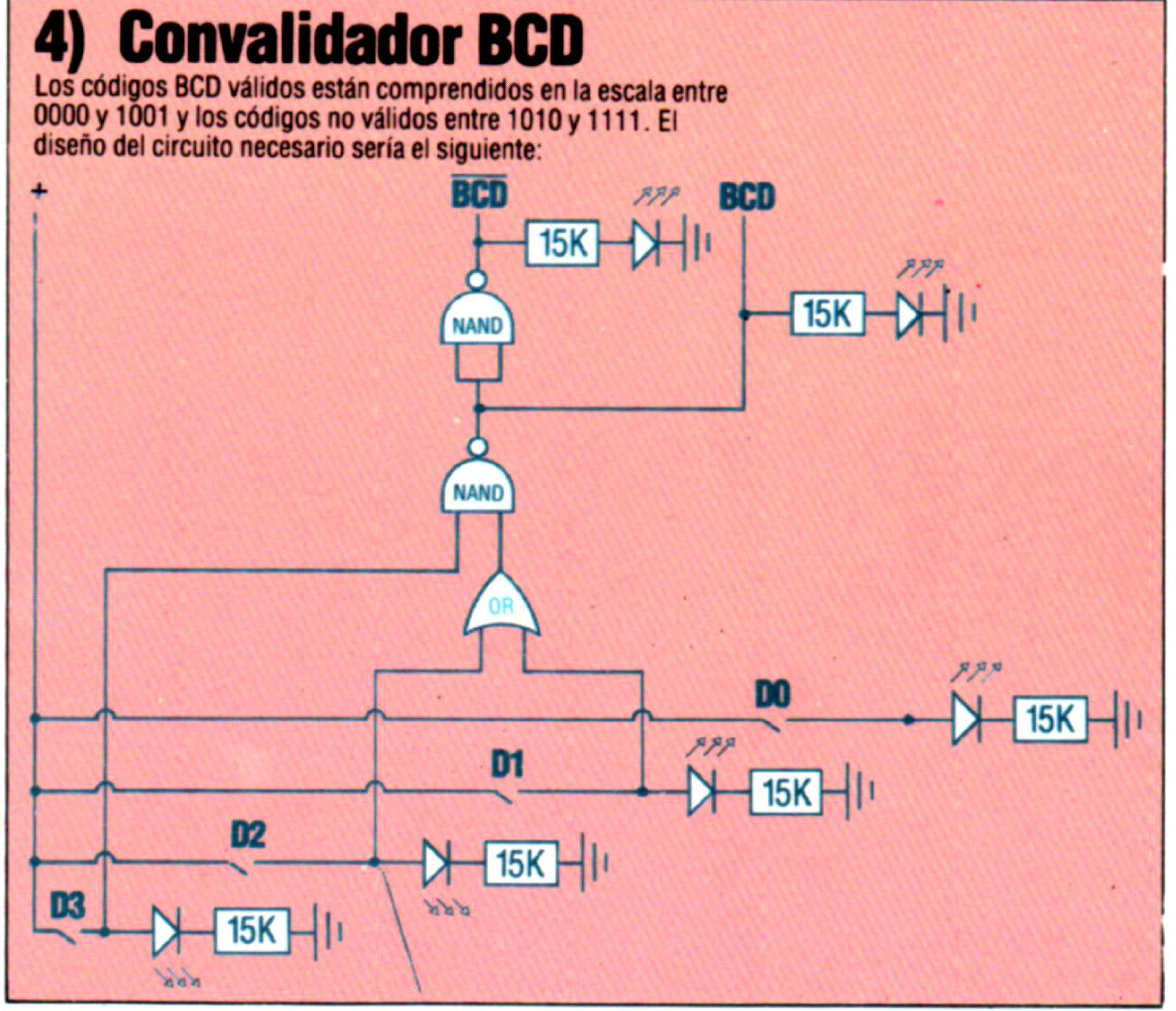
3) De decimal a binario

El convertidor de decimal a binario sólo requiere un circuito integrado: un chip TTL con cuatro puertas OR.



4) Convalidador BCD

Los códigos BCD válidos están comprendidos en la escala entre 0000 y 1001 y los códigos no válidos entre 1010 y 1111. El diseño del circuito necesario sería el siguiente:



Liz Dixon

Un portátil poderoso

El innovador micro Sharp PC-5000 cuenta con una visualización en cristal líquido incorporada y una memoria de burbuja

En cuanto a aspecto externo, el MS-DOS portátil de Sharp parece una pequeña máquina de escribir eléctrica. Pero bajo la tapa del PC-5000 se ocultan, entre otras configuraciones, una visualización en cristal líquido y una memoria de burbuja. Por otra parte, la carcasa del ordenador ha sido diseñada para dar cabida a su propia impresora opcional.

Los ordenadores portátiles asumen muchas formas, pero recientemente el estilo más popular ha sido el del portátil de regazo, como el TRS-80 Modelo 100 y el Epson HX-20. Estas máquinas incorporan una ligera pantalla LCD y pueden funcionar con bastante eficacia alimentadas solamente con pilas. El Sharp PC-5000 es la más cara de las máquinas de esta clase. Responde a esta tendencia de diseño y se adentra en un nuevo territorio al basarse en la memoria de burbuja en vez de en el almacenamiento en cassette.

Cuando hace algunos años se introdujo la idea de la memoria de burbuja, produjo una conmoción en la industria; pero la idea no cuajó en la medida que cabía esperar porque son pocas las compañías que han sido capaces de superar problemas relacionados con la velocidad y la fiabilidad. Sharp parece haber resuelto los problemas, porque su máquina de prueba demostró ser muy rápida y fiable. La memoria de burbuja requiere muy poca potencia y permite el almacenamiento de grandes cantidades de datos en un espacio diminuto. El principio implica almacenar datos en burbujas codificadas magnéticamente.

El PC-5000 tiene la apariencia de una máquina de escribir eléctrica pequeña y portátil. Al abrir la tapa queda a la vista un teclado esculpido tipo máquina de escribir con ocho teclas de función definida por el usuario y cuatro flechas de cursor colocadas elegantemente a través de la parte superior. La tapa está unida por medio de goznes y contiene la pantalla LCD. A pesar de ser bastante pesada, la tapa se mantiene en su sitio mediante un ensamblaje de trinquete, que permite moverla en varias posiciones para mayor comodidad. Arriba del teclado hay un panel con tres luces LED de aviso (potencia, poca pila, burbuja) y una ranura para un paquete de memoria burbuja.

Detrás de este panel hay una bandeja donde se sitúa la impresora opcional. Ésta es una caja rectangular que se inserta limpiamente en la bandeja. Es una impresora térmica que produce 37 caracteres por segundo. Sobre papel sensible al calor, la impresión ofrece muy buen aspecto, si bien el papel desluce en cierto sentido la calidad del documento.

El Sharp PC-5000 tiene 128 Kbytes de memoria para el usuario y 192 Kbytes de ROM, incluyendo BASIC Microsfot GW, la misma versión utilizada en el IBM PC. La CPU es el 8088 de Intel, nuevamente el mismo que el del PC, y el Sharp posee el MS-DOS como sistema operativo. El ordenador direcciona la burbuja como si ésta fuera las unidades de disco A y B. Sharp ofrece unidades de disco flexible gemelas que se pueden enchufar en la parte posterior del PC-5000. Estas unidades se direccionan como C y D y no pueden funcionar con pilas.

El Sharp viene con varios paquetes de software de Sorcim, incluyendo SuperCalc, SuperWriter y SuperComm, un programa de telecomunicaciones. Los programas vienen en un paquete burbuja y se seleccionan del menú del sistema pulsando una de las teclas de función. Los valores de las teclas de función pueden aparecer como etiquetas en la última línea de la pantalla. A Sharp se la conoce por su calidad en cuanto a diseño e ingeniería, y con el PC-5000 ha conseguido colocar un poderoso ordenador en un pequeño paquete.

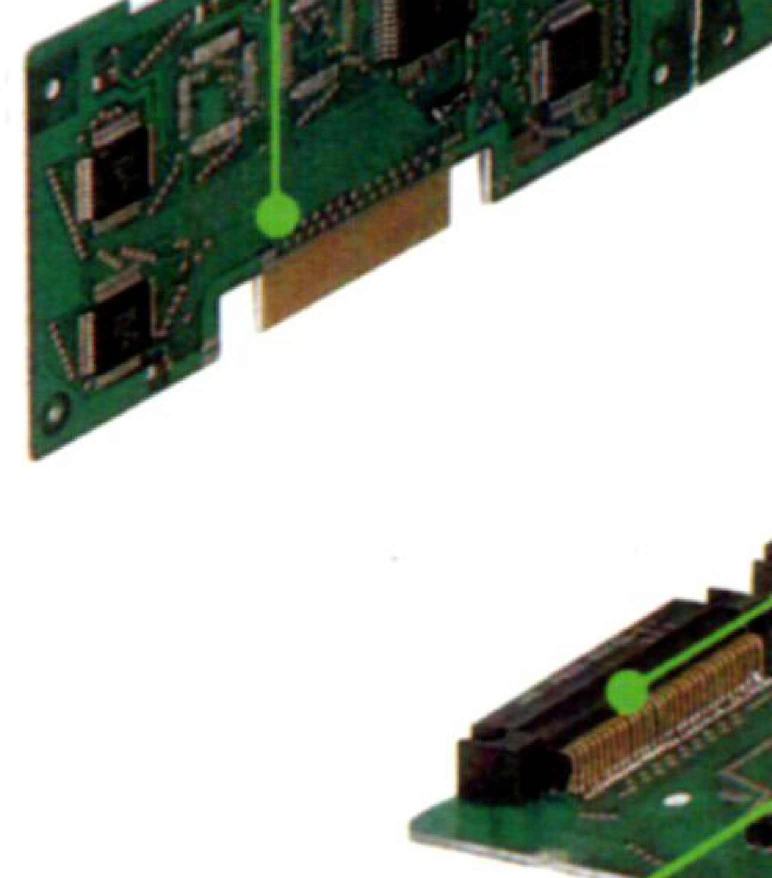


Ian McKinnell

Unidades gemelas de disco flexible
Esta unidad contiene dos unidades de disco de 320 K bajo MS-DOS. Se conecta en la parte posterior del PC-5000, pero no puede funcionar dependiendo de las pilas de la máquina. Al software escrito para otras máquinas MS-DOS se les debe formatear para la visualización en 8 líneas



Impresora opcional
Mirando el PC-5000 desde arriba se puede ver la impresora térmica opcional anidada en la carcasa. La impresora encaja exactamente en una pequeña bandeja y se conecta al tablero del sistema mediante un cable plano. Con sus 37 caracteres por segundo la impresora es veloz, y produce una salida de gran calidad sobre papel sensible al calor



Controlador de memoria de burbuja
Esta ficha actúa como un controlador de unidad de disco, excepto en que controla la entrada y la salida desde la ranura del paquete burbuja

Tablero ROM
Este tablero posee 128 K de ROM incluyendo MS-DOS, generadores de caracteres para la visualización y la impresora, algunas comunicaciones y E/S del sistema y el PISCS. El PISCS es un chip que traduce las señales del paquete burbuja a un lenguaje que el MS-DOS entiende. Con la ayuda de este chip, el MS-DOS lee los paquetes burbuja como las unidades de disco A y B

Ranuras para ampliación
Estas ranuras retienen los paquetes de RAM o el software retenido en cartuchos ROM



RAM dinámica de 128 K

Ranura para paquete burbuja

Impresora térmica

Circuitos integrados híbridos
Estos circuitos integrados de cerámica albergan los activadores para la puerta RS232, el conector de la unidad de disco y la fuente de alimentación eléctrica, entre otros

CPU 8088

Microordenador de un único chip CMOS
Este chip controla la operación de todo el sistema, y cierra los chips 8088, de E/S, etc., cuando no se los está utilizando para preservar energía

Ian McKinnell

SHARP PC-5000

DIMENSIONES
300 × 318 × 90 mm

PESO
5,7 kg incluyendo la impresora

CPU
8088 de Intel de 16 bits y CMOS de 8 bits

MEMORIA
128 K de RAM
192 K de ROM

PANTALLA
LCD de 80 caracteres por 8 líneas
Resolución de gráficos de 640 × 80
Caracteres internacionales y para gráficos incorporados

INTERFACES
Cassette, unidad de disco externa, salida para modem, RS232, adaptador para corriente CA, puertas para ROM-RAM

LENGUAJES DISPONIBLES
BASIC Microsoft GW

TECLADO
Estilo máquina de escribir, estándar, de 57 teclas, con 8 teclas de función, teclas para control del cursor, otras tres teclas especiales. Los valores de las teclas de función se definen mediante software y pueden aparecer en la última línea de la pantalla

DOCUMENTACION
Los manuales que se proporcionan incluyen información relativa al montaje del PC-5000 y a la utilización del MS-DOS, y son bastante buenos. El manual de BASIC lo ha escrito Microsoft y es excesivamente técnico para el usuario recién iniciado

VENTAJAS
El 5000 ofrece la mayoría de las facilidades de una máquina de tamaño natural. La impresora incorporada y los cartuchos burbuja, en particular, la colocan por delante de sus rivales

DESVENTAJAS
Existen pocos puntos débiles si uno considera los desarrollos que Sharp ha llevado a cabo. No obstante, el 5000 es pesado para sus dimensiones y más caro que otras máquinas de regazo. Si usted está acostumbrado a los discos flexibles, encontrará un tanto lentas las memorias de burbuja

Último informe

Completamos nuestra serie de capítulos sobre software de gestión exponiendo otros métodos para analizar los datos contables



Ian McKinnell

Los responsables del control de almacén han de contar con información suficiente no sólo de un artículo en particular. Necesitan conocer todos los aspectos del stock actual y de los movimientos de existencias. Hay dos maneras de lograrlo: por medio de consultas en pantalla o a través de informes impresos. El menú de consultas del programa Dragon contiene varias opciones con el tipo de datos disponible y cómo se presentarán.

Una facilidad esencial de consulta o informe es la de los artículos con escaso movimiento de venta. Dado que todas las operaciones son fechadas cuando se digitan en el sistema, el programa dispone de antemano de la información necesaria para generar un informe sobre artículos de movimiento lento. Le bastará con indagar en el archivo que contiene el historial de todas las operaciones y comparar datos. Como cada empresa tiene su propio concepto de "lentitud" (lo que para una es movimiento lento podría ser un ritmo excelente para otra), el informe ha de permitir que el usuario defina los términos.

El sistema Dragon satisface ampliamente este punto. Digitando una fecha determinada (p. ej., 150484, por 15 de abril de 1984), el usuario da al sistema una clave para comenzar la búsqueda. El programa lee y luego imprime todos los artículos sin movimiento de ventas. Esto proporciona una facilidad para informes muy útil, porque se puede generar cualquier número de informes de movimiento lento simplemente proporcionándole al programa una fecha diferente. La única precaución que ha de tomar el usuario es que la fecha esté comprendida dentro de los límites del historial de transacciones del archivo. De no haber ningún artículo de movimiento lento que caiga dentro de la fecha especificada, el mensaje en pantalla del programa Dragon dirá: "NINGUN ARTICULO SEGUN CRITERIO DE SELECCION".

Existe una limitación para este tipo de informes. Sólo detecta artículos de las existencias que no hayan sido objeto de absolutamente ninguna transacción. Pero está claro que en algunos casos un usuario podría muy bien pensar que una determinada línea de existencias del cual se han producido dos ventas en seis meses se debería considerar como de movimiento lento.

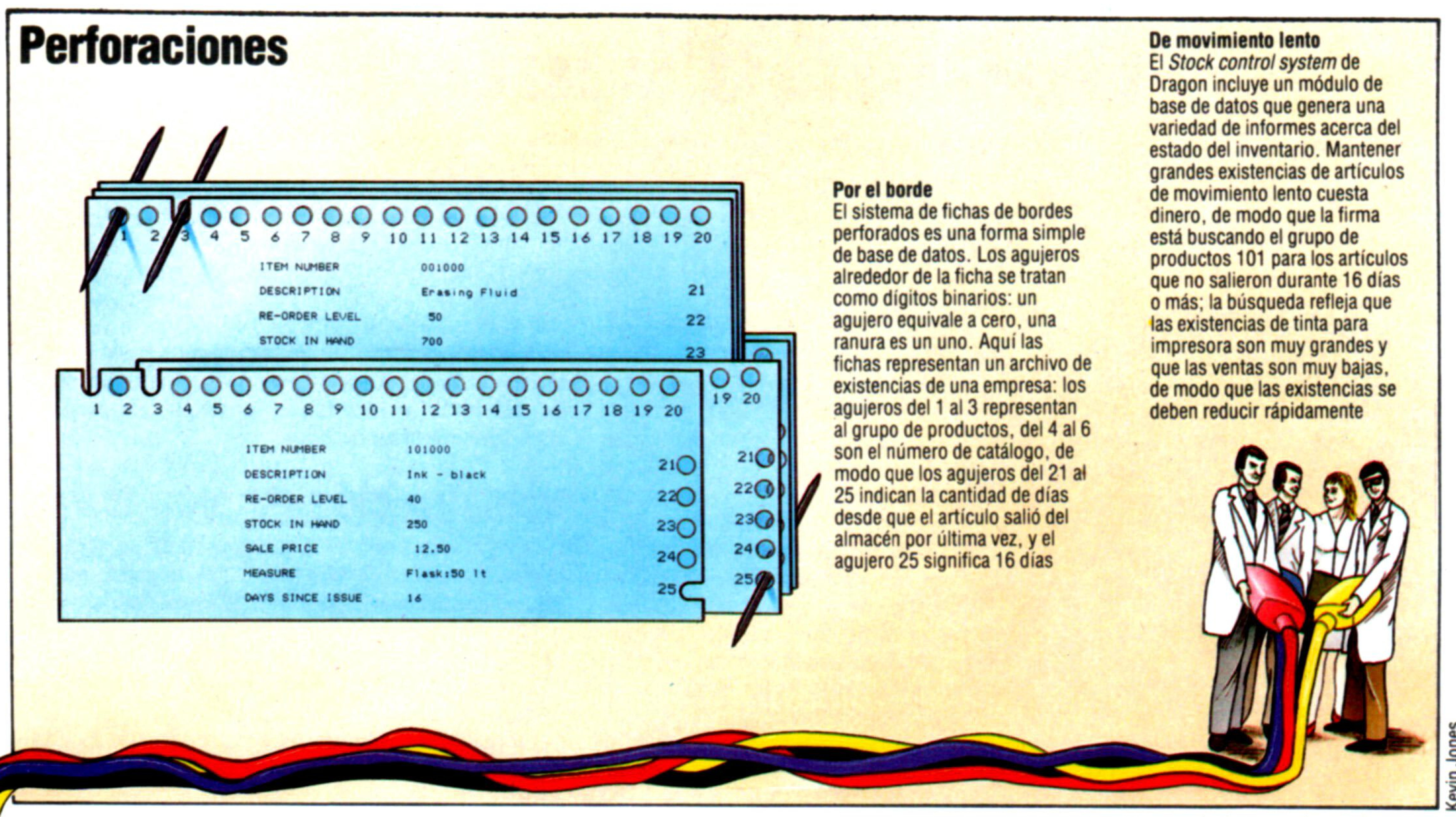
Un sistema más sofisticado proporcionaría una mayor flexibilidad. Hay dos formas de conseguir esto. El sistema podría ofrecer un criterio de selección adicional distinto del de la fecha, como especificar el número de transacciones por debajo del cual un artículo se imprimirá en el informe de movimiento lento. O bien, hacer que el mismo sistema lea y compare los movimientos de todas las partículas, listando, pongamos por caso, los 50 artículos más lentos, luego los siguientes 50 artículos más lentos y así sucesivamente.

Visualización en pantalla

La pantalla visualiza cada artículo por separado. Por consiguiente, si hay un gran número de artículos en existencias de movimiento lento, tener que pasar las páginas de la lista nos llevaría mucho tiempo. En ese caso, por lo tanto, un informe impreso podría muy bien ser una alternativa mejor que una visualización en pantalla, porque es más rápido y más fácil de explorar. El diseñador de sistemas ha de tener esto en consideración cuando se compone o "especifica" un programa de gestión como un paquete de control de existencias.

La visualización del movimiento de existencias contiene una gran parte de información adicional que es importante para el usuario. Refleja el valor de las existencias, lo que le indica al usuario cuánto capital hay comprometido en un almacén o en un

Perforaciones



Por el borde
El sistema de fichas de bordes perforados es una forma simple de base de datos. Los agujeros alrededor de la ficha se tratan como dígitos binarios: un agujero equivale a cero, una ranura es un uno. Aquí las fichas representan un archivo de existencias de una empresa: los agujeros del 1 al 3 representan al grupo de productos, del 4 al 6 son el número de catálogo, de modo que los agujeros del 21 al 25 indican la cantidad de días desde que el artículo salió del almacén por última vez, y el agujero 25 significa 16 días

De movimiento lento
El *Stock control system* de Dragon incluye un módulo de base de datos que genera una variedad de informes acerca del estado del inventario. Mantener grandes existencias de artículos de movimiento lento cuesta dinero, de modo que la firma está buscando el grupo de productos 101 para los artículos que no salieron durante 16 días o más; la búsqueda refleja que las existencias de tinta para impresora son muy grandes y que las ventas son muy bajas, de modo que las existencias se deben reducir rápidamente

Kevin Jones

depósito. Muestra el uso promedio por mes, la cantidad en existencia y la fecha de la última distribución. A partir de esta información, el usuario podrá elaborar una política para renovar las existencias, ofreciéndolas quizá con grandes descuentos.

El principio de elaborar informes de acuerdo a un criterio de selección (la fecha, p. ej.) es fundamental cuando se trata de analizar datos de transacciones e informar sobre ellos. Obviamente, el sistema necesita ser capaz de listar todas las transacciones de todos los artículos. Pero también debería indicar qué ha sucedido con secciones de las existencias entre fechas especificadas.

El programa Dragon lo consigue pidiéndole al usuario que dé entrada a los valores superiores e inferiores de una gama de artículos de las existencias y las fechas de delimitación (p. ej., entre las fechas 010184 y 010484).

Además de contemplar las transacciones habidas con una gama específica de artículos, al usuario podría interesarle, asimismo, una descomposición, con los artículos agrupados de acuerdo al tipo de transacción, o cifras de ventas, o ajustes de existencias, etc. El programa Dragon poseen un menú con tres opciones para las fechas seleccionadas (a saber: sólo transacciones del período actual; todas las transacciones; y transacciones dentro de un período especificado), y luego otro submenú para descomponer la selección por tipos de transacciones. Este submenú es similar al menú de tipo de transacciones que describimos en la página 672, pero también incluye una nueva opción, `99 ALL TYPES` (todo tipo), que permite una salida impresa general de todas las transacciones dentro de los parámetros de datos especificados.

El programa permite a los usuarios incluir los artículos individuales a grupos determinados. Así, los usuarios podrán revisar los grupos de productos y sus descripciones. También pueden saber cuáles son las existencias disponibles para cada grupo determinado de productos. Los informes basados en pantalla son útiles, pero no son informes permanentes. Con el fin de proporcionar un registro de "salida impresa", la facilidad de informes duplica en muchos sentidos la facilidad de consultas que ofrece este programa.

Hay una cantidad de razones por las cuales el programa Dragon es un sistema de control de existencias bastante restringido, a pesar de sus detalladas configuraciones analíticas. El sistema está diseñado como un sistema de existencias "en solitario". No puede conectarse con otros programas para aplicaciones de gestión que podrían utilizar la información de los archivos de este programa. No hay ninguna facilidad para señalar las existencias según pedidos. Una de las preguntas más importantes que a los usuarios les agradaría hacer: "¿Tengo suficientes artículos en existencia para atender tales pedidos?" Cuando a usted se le amontonan varias órdenes de clientes por un número de artículos diferentes, es muy difícil llevar de forma manual el registro de las demandas sobre las existencias.

En toda esta serie de capítulos nos hemos concentrado en la utilización de los ordenadores personales para el procesamiento de datos en pequeñas empresas. A estas alturas debería estar claro que a pesar de que estos paquetes podrían ofrecer supuestamente sistemas integrados, con frecuencia se concentran en un aspecto del negocio (p. ej., el movimiento de caja) en detrimento de otros, como, por ejemplo, procesamiento de pedidos, inventario, conformidad. Quizá la mejor oferta de estos paquetes es la de proporcionarles información suplementaria a los directores de empresas.

A pesar de estos inconvenientes, los paquetes de gestión para ordenadores personales pueden mejorar sustancialmente la eficacia de la contabilidad y la organización de una pequeña empresa. Elegidos y empleados con cuidado, dichos paquetes pueden resultar verdaderamente útiles.

Acción en alta mar

Comenzaremos a estudiar la capacidad de gráficos del Commodore 64. Esto nos permitirá construir y analizar un juego muy atractivo

Una de las configuraciones más atractivas del Commodore 64 es su capacidad para escribir juegos de estilo recreativo en BASIC. Eso es posible gracias a la capacidad que tiene la máquina para manipular sprites: formas de alta resolución que el usuario puede definir y manipular fácilmente en la pantalla. En la memoria del Commodore hay registros especiales que controlan los atributos de los sprites, como su color, su tamaño y su posición en la pantalla. Colocando (POKE) números en estos registros, el programador puede controlar la acción con gran facilidad. El juego *Subhunter* que vamos a diseñar como proyecto central de esta serie hará uso de cuatro de los ocho sprites disponibles de la máquina. Estos sprites representarán al barco, al submarino, a las cargas de profundidad disparadas desde el barco y a una explosión.

De modo que el juego se puede ir elaborando a lo largo de sucesivos capítulos, escribiendo cada una de las secciones del programa como breves subrutinas que serán llamadas para su utilización dentro del bucle principal del programa. Así es más sencillo rastrear errores y ampliar el programa.

El concepto de este juego resultará familiar a muchos lectores. El jugador tiene bajo su control un barco que se encuentra a la caza de submarinos. Éstos atraviesan la pantalla a una profundidad y velocidad variables y el barco les arroja cargas de profundidad. El marcador del jugador aumenta cada vez que toca a un submarino, calculándose el valor de cada uno de éstos a tenor de su profundidad y su velocidad. Naturalmente, las marcas más elevadas se consiguen alcanzando a submarinos que se mueven con mayor velocidad y a mayor profundidad. No obstante, si el submarino escapa se resta del marcador del jugador el valor de aquél. El barco se controla desde el teclado, utilizando las teclas Z y X para el movimiento horizontal hacia izquierda y derecha. Las cargas de profundidad se disparan mediante la tecla M. En la pantalla se visualiza un reloj, que permite jugar sólo durante tres minutos. Transcurrido este tiempo se le pregunta al jugador si desea jugar otra vez y se le ofrece un registro de la marca más alta obtenida.

La regla de oro a observar cuando se diseña un juego de acción en BASIC es que el bucle principal del programa sea lo más corto posible. El programa *Subhunter* se vale de las subrutinas para llevar a cabo la mayoría de las funciones del juego. Las únicas funciones que se controlan directamente desde dentro del bucle principal del programa son: actualización del reloj, aceptación de una entrada desde el teclado, desplazamiento del barco y desplazamiento del submarino.

Este diseño de programa es lo suficientemente general como para poderse aplicar a cualquier marca de ordenador, pero la programación detallada variará en función de las características individuales del que se esté utilizando. En esta sección del proyecto analizaremos la rutina que crea una visualización en pantalla.

Visualización en pantalla

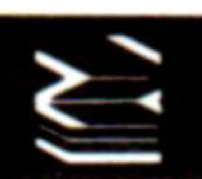
En el Commodore 64 hay dos formas de imprimir los caracteres en la pantalla. Una consiste en emplear la sentencia PRINT y la otra en utilizar POKE para colocar números en las zonas de la memoria que retienen información relativa a la visualización. Utilizaremos ambos métodos para la creación del telón de fondo de nuestro juego.

La pantalla Commodore se compone de 25 filas y 40 columnas o espacios para caracteres. En otras palabras, en la pantalla hay 1 000 lugares donde se puede colocar un carácter. Cada posición de la pantalla tiene dos números relacionados con ella. El primero es un número de código de pantalla que le dice al ordenador qué carácter visualizar en esa posición. El segundo es un número de color que le indica al ordenador de qué color debe ser el carácter visualizado. Hay dos bloques de memoria, cada uno de los cuales consta de 1 000 posiciones: uno para retener información relativa al código de pantalla y otro para retener información relativa al color de cada posición de la pantalla. La zona que retiene los códigos de pantalla se denomina *memoria de pantalla* y va desde la posición 1024 a la 2023. La memoria de color va desde la 55296 a la 56295.

Cada carácter posee su propio código de pantalla y éstos están listados en el manual para el usuario. En el Commodore 64 hay 16 colores. Los códigos de color son:

0	negro	8	naranja
1	blanco	9	marrón
2	rojo	10	rojo claro
3	cyan	11	gris 1
4	púrpura	12	gris 2
5	verde	13	verde claro
6	azul	14	celeste
7	amarillo	15	gris 3

Además de estas zonas de la memoria, hay otras dos posiciones que son de especial interés. La posición 53280 controla el color del borde y la 53281 controla el color de la pantalla. Para diseñar el paisaje marino que necesitamos para nuestro juego, las seis filas superiores de la pantalla serán de color celeste para representar el cielo, mientras que el resto de la pantalla y el borde serán azul oscuro para evocar el mar (excepto las dos filas inferiores, que representarán el lecho marino).



Para establecer el color de la pantalla en celeste y el color del borde en azul oscuro, se requiere el siguiente par de órdenes POKE:

```
POKE53281,14:POKE53280,6
```

La séptima fila de la pantalla comienza en la posición 1264. La segunda fila por la parte inferior comienza en la posición 1944. Podemos colorear el mar colocando (POKE) el código de pantalla para un espacio invertido (un carácter espacio bloqueado) en las posiciones 1264 a 1943, al tiempo que se coloca (POKE) 6 en las correspondientes posiciones de memoria de color. Existe una forma sencilla de conectar una posición de memoria de color con una correspondiente posición de memoria de pantalla: sumándole simplemente 54272 a la posición de memoria de pantalla.

El código de pantalla para un carácter espacio es 32. El código de pantalla para un carácter invertido se puede calcular sumando 128 al código de pantalla normal, de modo que el código para un espacio invertido es 32 + 128 = 160. Las siguientes líneas del programa se valen de un bucle FOR...NEXT simple para colorear el mar:

```
FOR I = 1264 TO 1943
POKE I,160:POKE I + 54272,6
NEXT I
```

El lecho marino se compone de dos filas de un espacio cuadrado con un código de pantalla de 102, de color marrón. Nuevamente, un simple bucle FOR...NEXT se encargará de esto:

```
FOR I = 1944 TO 2023
POKE I,102:POKE I + 54272,9
NEXT I
```

La sentencia PRINT también es un método eficaz para producir visualizaciones en pantalla. El color y el posicionamiento del cursor se pueden controlar desde ella mediante los caracteres de control especiales de Commodore o bien utilizando códigos CHR$. Utilizaremos este último método, porque es más fácil de leer en los listados de programas. En un apéndice del manual del usuario hay una relación completa de los códigos CHR$. A nosotros nos interesan aquellos que se refieren al color y al posicionamiento del cursor:

CHR$(5)	color blanco
CHR$(144)	color negro
CHR$(147)	limpia la pantalla y posiciona el cursor en el ángulo superior izquierdo
CHR$(19)	posiciona el cursor en el ángulo superior izquierdo
CHR$(17)	desplaza el cursor un lugar hacia ABAJO
CHR$(145)	desplaza el cursor un lugar hacia ARRIBA
CHR$(157)	desplaza el cursor un lugar hacia la IZQUIERDA
CHR$(29)	desplaza el cursor un lugar hacia la DERECHA

Como parte de nuestra rutina de organización de la pantalla se ha de imprimir MARCADOR y MARCADOR MAX en la línea superior de la pantalla. CHR$(19) asegurará que el cursor esté al comienzo de la línea superior. La siguiente orden imprime el marcador inicial en negro:

```
PRINT CHR$(19);CHR$(144);"MARCADOR 000"
```



El Subhunter en acción
El programa que estamos desarrollando en esta serie es el clásico juego, con gráficos por ordenador, de caza de submarinos con cargas de profundidad. Se utilizan los gráficos sprite del Commodore 64 para dar una animación continua al submarino y al barco

MARCADOR MAX también se posiciona en la línea superior, pero sobre el lado derecho. La función SPC permite insertar un número de espacios. La orden PRINT ahora se puede alterar para que incluya MARCADOR MAX:

```
PRINT CHR$(19);CHR$(144);
"MARCADOR 000";SPC(16);"MARCADOR MAX
  000"
```

La rutina de preparación de la pantalla formará una subrutina para el programa principal, empezando en la línea 1000. También se incluye una orden POKE que hace que todas las teclas se repitan cuando se pulsen. Esto se utilizará cuando analicemos la rutina de control del teclado. Esta subrutina se puede comprobar mediante las siguientes líneas:

```
10 GOSUB 1000: REM PREPARACION PANTALLA
20 END
1000 REM ****PREPARACION PANTALLA****
1010 PRINT CHR$(147):REM LIMPIAR PANTALLA
1020:
1030 REM**COLOR MAR**
1040 POKE53281,14:POKE53280,6
1050 FOR I = 1264TO1943
1060 POKE I,160:POKE I + 54272,6
1070 NEXT
1080:
1090 REM**FONDO MAR**
1100 FOR I = 1944TO2023
1110 POKE I,102:POKE I + 54272,9
1120 NEXT
1130 POKE 650,128:REM REPETIR TECLAS
1140:
1150 REM**MARCADOR**
1160 PRINTCHR$(19);CHR$(144);"MARCADOR
     000";SPC(16);"MARCADOR MAX 000"
1170 RETURN
1180:
1190:
```

Después de que se ha dado entrada a la rutina, es una buena idea guardarla en cinta o en disco antes de ejecutarla.

Instrucciones para el contador

Aprender a utilizar de manera adecuada etiquetas y bucles es fundamental para todo aquel que se proponga programar eficientemente en lenguaje máquina

Los bucles y las bifurcaciones condicionadas se ejecutan en lenguaje assembly utilizando los flags del registro indicador de estado con objeto de probar la condición del acumulador, y las instrucciones del salto relativo para modificar el flujo de control del programa. En la creación de tablas de datos se combinan a un tiempo estas estructuras y la modalidad de direccionamiento indexada.

Antes de que podamos utilizar provechosamente las diversas modalidades de direccionamiento de la CPU (en especial las direcciones indexadas), debemos primero ser capaces de escribir un bucle. Sin esta estructura fundamental estamos en una situación bastante similar a la de un programador de BASIC que sabe mucho de matrices pero que ignora la orden FOR...NEXT. En lenguaje assembly no hay estructuras automáticas como FOR...NEXT (aunque hay una instrucción para el Z80 que se aproxima mucho a la misma), pero podemos construir bucles de tipo IF...THEN GOTO... Éstos requieren instrucciones que tomen decisiones o expresen condiciones y, efectivamente, cambien el orden por el cual se obedecen las órdenes en el programa.

La toma de decisiones en lenguaje assembly se centra en los flags del indicador de estado del procesador. Tales flags, o banderas, muestran los efectos en el acumulador de la última instrucción ejecutada y en algunas ocasiones se denominan *flags de condición*. Todos ellos se pueden utilizar en la toma de decisiones, pero por el momento necesitamos considerar sólo dos de los mismos: el flag de cero (Z) y el de arrastre (C).

El estado de estos flags se utiliza para decidir si el procesador ejecuta la siguiente instrucción del programa, o si salta a alguna otra instrucción de éste. El procesador decide si continuar o saltar, bien cambiando o bien aceptando la dirección que contiene su contador del programa. Este registro siempre contiene la dirección de la siguiente instrucción en lenguaje máquina a obedecer. Cuando el procesador empieza a ejecutar una instrucción, carga en el contador del programa el opcode de la instrucción contenido en el byte cuya dirección es indicada por el contador. La dirección del registro se incrementa en razón del número de bytes de la instrucción, de modo que el contador del programa señala entonces el opcode de la siguiente instrucción. Si la que está en curso hace que el contador del programa señale a una dirección de algún otro lugar del programa, entonces se genera efectivamente un salto.

En el 6502, la instrucción BEQ hace que el contador del programa cambie si el flag de cero está activado. BCS es la instrucción equivalente en el caso de que el flag de arrastre esté activado. En el Z80, estas instrucciones son JR Z y JR C, respectivamente. Estos cuatro opcodes se denominan *instrucciones de bifurcación*, porque representan un punto de bifurcación en el flujo de control del programa. Su operando es un número de un único byte, que se le suma a la dirección del contador del programa con objeto de indicar una nueva dirección. Veamos a continuación lo que sucede cuando se ejecuta el siguiente programa:

ORG $5E00			
6502		Z80	
5E00	ADC #$34	ADC	A,$34
5E02	BEQ $03	JR	Z,$03
5E04	STA $5E20	LD	($5E20),A
5E07	RTS	RET	

Si la instrucción ADC en $5E00 produce en el acumulador un resultado cero (poco probable, pero posible, como ahora veremos), entonces las instrucciones BEQ y JR Z en $5E02 harán que se le sume $03 al contenido del contador del programa. La siguiente instrucción a ejecutar, por lo tanto, será RTS, la instrucción de retorno en $5E07, saltándose la instrucción en $5E04.

A primera vista esto podría parecer erróneo. Después de todo, si la instrucción en $5E02 hace que se le sume $03 al contador del programa, la dirección almacenada será sin duda $5E05. En este punto resulta imprescindible tener siempre presente que el contador del programa lo que señala es la *siguiente* instrucción a ejecutar y no la instrucción que se está realizando en cada momento. Por consiguiente, cuando comience la ejecución de la instrucción en $5E02, el contador del programa contendrá la dirección $5E04: la posición de la siguiente instrucción. Si a $5E04 se le suma $03, el resultado será $5E07, la dirección de la instrucción siguiente.

Vale la pena recalcar que el procesador no sabe verificar si las direcciones que se le indican son las correctas. Si, por descuido, cambiáramos el desplazamiento de la instrucción a $02, entonces el contador del programa se incrementará (si el acumulador contiene cero) en $02, y el procesador considerará $5E06 como la dirección del opcode de la siguiente instrucción. En nuestro programa correcto, $5E06 contiene el valor $5E, que es el byte *hi* del operando de la instrucción contenida en $5E04. El procesador no puede evaluar si es la instrucción correcta o no. Para él, $5E es un opcode válido y procederá a eje-

cutarlo, tomando los bytes que siguen a $5E06 como los operandos de la instrucción. A resultas de lo cual el programa desbarrará. Calcular mal, como aquí, los desplazamientos es uno de los errores más comunes que se producen en la programación máquina.

Sin embargo, en la programación con assembly calcular los desplazamientos de salto no constituye ningún problema, porque el programa ensamblador lo puede hacer por nosotros. Y así, en vez de escribir un byte hexa como operando de la instrucción de bifurcación, damos la dirección simbólica de la instrucción a la cual saltar. Con esto resulta mucho más fácil seguir el programa en lenguaje assembly. El ensamblador decodifica la dirección simbólica otorgándole una dirección absoluta, calcula el desplazamiento necesario para llegar a la dirección y escribe ese desplazamiento en la instrucción en lenguaje máquina. La dirección simbólica se denomina *etiqueta* y es análoga a un número de línea de un programa en BASIC.

Analicemos con más detalle el empleo de las etiquetas. Una etiqueta es una serie alfanumérica escrita al principio de una instrucción en lenguaje assembly. El programa ensamblador la trata como un símbolo de dos bytes que representa la dirección del primer byte de la instrucción. Veamos ahora de qué forma quedaría el programa dado aplicando esta nueva técnica:

ORG $5E00				
	6502		Z80	
5E00	ADC	#$34	ADC	A,$34
5E02	BEQ	EXIT	JR	Z,EXIT
5E04	STA	$5E20	LD	($5E20),A
5E07 EXIT	RTS		RET	

Ahora la instrucción en $5E02 es como si dijera: "si (IF) el valor del acumulador es cero, entonces ir (THEN GOTO) a la dirección representada por la etiqueta EXIT". Vemos que en comparación con la versión previa, hemos mejorado la legibilidad del programa y hace que disminuyan en buena parte las posibilidades de calcular mal el destino del salto.

Ahora ya estamos en disposición de poder utilizar las etiquetas y las instrucciones de bifurcación para crear un bucle:

ORG $5E00				
	6502		Z80	
5E00 START	ADC	#$34	ADC	A,$34
5E02	BNE	START	JR	NZ,START
5E04	STA	$5E20	LD	($5E20),A
5E07 EXIT	RTS		RET	

Observe aquí la utilización de la nueva etiqueta, START, así como las nuevas instrucciones de bifurcación: BNE, que significa "bifurcar si el acumulador no es igual a cero" (**B**ranch *if* **N**ot **E**qual); y JR NZ, que significa "saltar si el acumulador no es igual a cero" (**J**ump *if* **N**ot **Z**ero). Consideremos el efecto que tendrá este código. El programa primero sumará $34 al acumulador. Si el resultado no es igual a cero, entonces el programa se bifurca hacia atrás, hasta $5E00 (la dirección donde se colocó la etiqueta START). $34 se le volverá a sumar al acumulador, y el resultado decidirá si se produce de nuevo ese retroceso. El bucle se repetirá una y otra vez mientras se satisfaga la condición negativa de la bifurcación. Cuando el contenido del acumulador sea igual a cero como resultado de una instrucción ADC, entonces no se producirá la bifurcación en $5E02 y se ejecutará seguidamente la instrucción contenida en $5E04.

Esto es exactamente como un bucle IF...THEN GOTO... de BASIC. Sólo que quizá usted no acabe de creer cómo el acumulador puede convertirse alguna vez en cero. Porque, lo que sí está claro es que ¡se está incrementando en razón de $34 cada vez que se ejecuta el bucle! ¿Cómo es posible que dé cero alguna vez? La respuesta se basa en el hecho de que el acumulador es en definitiva un registro de un único byte, y si la suma da como resultado un número de dos bytes, entonces se activará el flag de arrastre en el indicador de estado del procesador y el acumulador sólo retendrá el byte *lo* del resultado. Si en un determinado momento el acumulador contiene $CC, por ejemplo, entonces al sumarle $34 se obtendrá el número de dos bytes $0100. Se activará el flag de arrastre y el acumulador retendrá el byte *lo* de este resultado: $00. He aquí cómo el contenido del acumulador será cero a resultas de lo cual el flag de cero se activará.

Teniendo en mente este resultado, podríamos reescribir el programa para utilizar una condición de bifurcación diferente, incorporando el estado del flag de arrastre en lugar del estado del flag de cero.

ORG $5E00				
	6502		Z80	
5E00 START	ADC	#$34	ADC	A,$34
5E02	BCC	START	JR	NC,START
5E04	STA	$5E20	LD	($5E20),A
5E07 EXIT	RTS		RET	

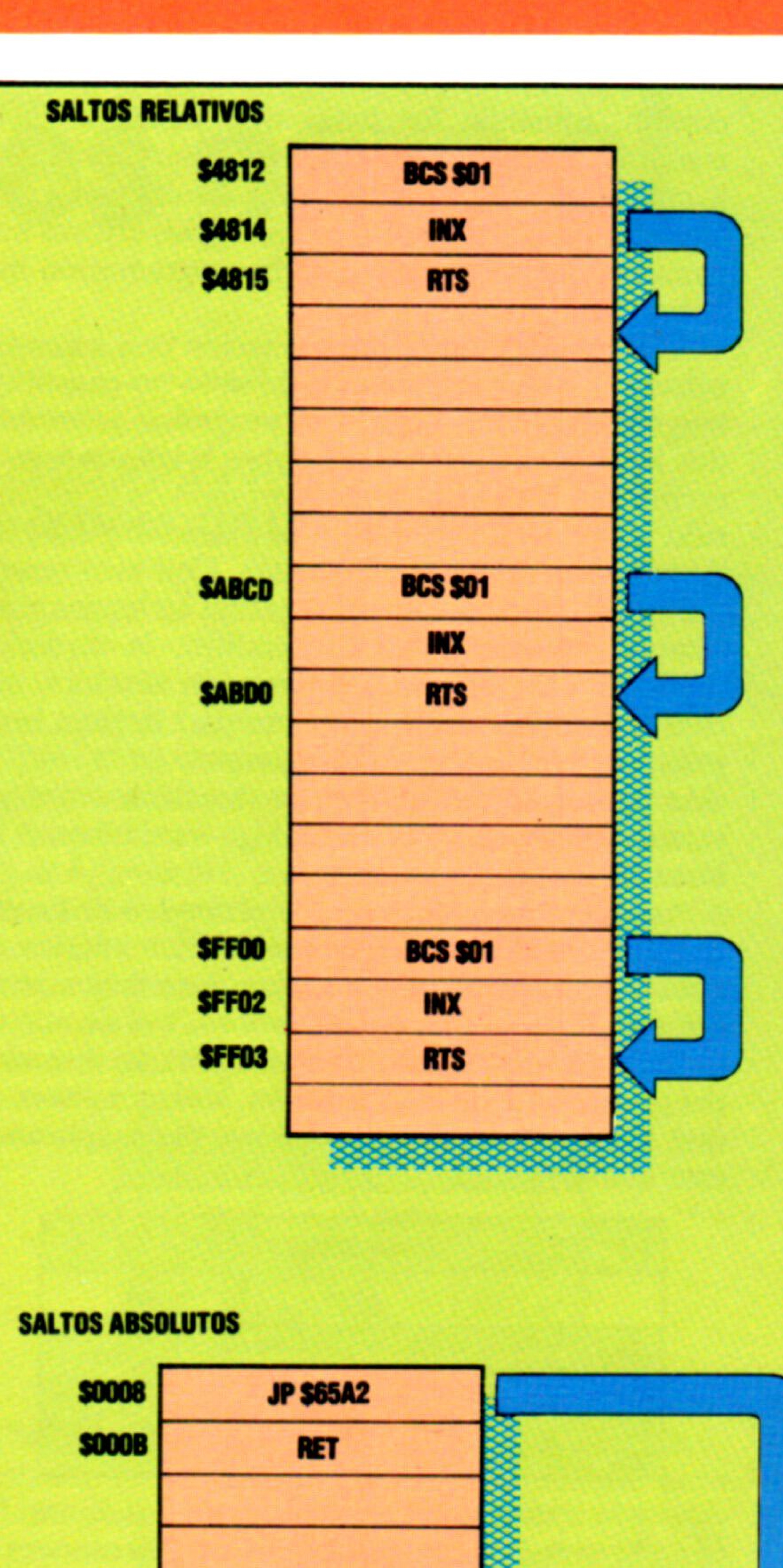
En esta versión, la instrucción en $5E02 significa: "si el flag de arrastre se activa, bifurcar a START". Enseguida que el resultado de sumarle $34 al acumulador sea mayor que $FF, el flag de arrastre quedará activado y dejará de producirse la bifurcación hacia atrás hasta la dirección START.

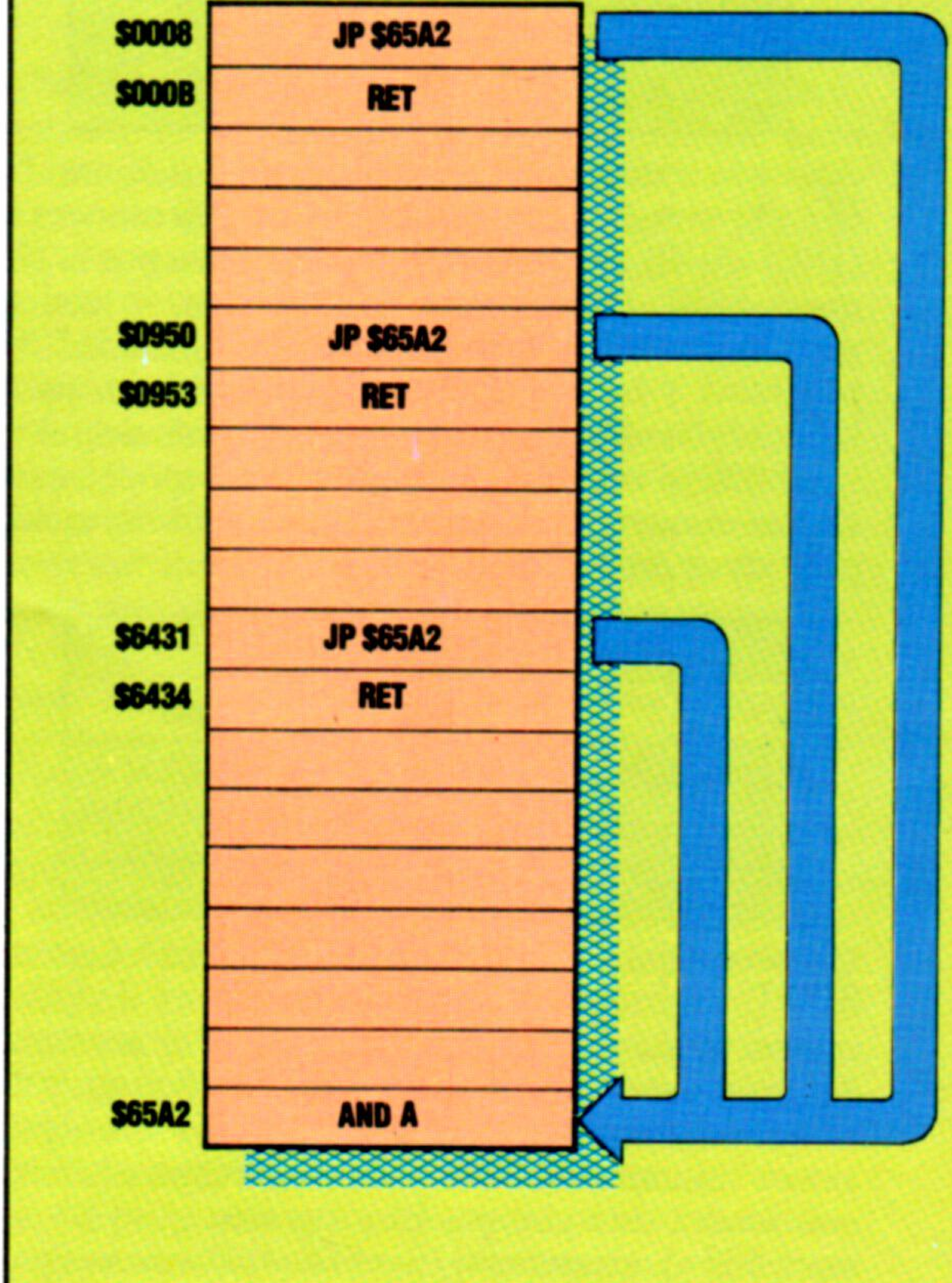
Contadores de bucles

Puede que le parezca bastante limitada esta posibilidad de bifurcación según el estado del flag de arrastre o de cero, pero nos permite un amplio margen de maniobra, tal como veremos enseguida. Lo que sí que deberá estar echando en falta es un *contador de bucle*. Es posible que necesite, por ejemplo, contar el número de veces que se efectúa un bucle antes de que se produzca la condición de salida, o que desee realizar una salida del bucle al cabo de un número dado de iteraciones. Lo primero se alcanza fácilmente con un registro de índice de la CPU que guarde el contador, más una instrucción de incremento para actualizarlo:

6502		
0000	ORG	$5DFD
5DFD	LDX	#$00
5DFF START	INX	
5E00	ADC	#$34
5E02	BCC	START
5E04	STX	$5E20
5E07 EXIT	RTS	

Z80		
0000	ORG	$5DFA
5DFA	LD	IX,$0000
5DFE START	INC	IX
5E00	ADC	A,$34
5E02	JR	NC,START
5E04	LD	($5E20),IX
5E08 EXIT	RET	





El programa presenta ahora las novedades que pasamos a explicarle. En primer lugar, las instrucciones que escribíamos al principio del programa requieren una nueva dirección ORG. El efecto de tales instrucciones sabemos que es muy similar tanto en el 6502 como en el Z80, pero sus longitudes son diferentes, de modo que las direcciones de su posición ya no son las mismas en las dos versiones del programa.

En segundo lugar, se ha utilizado una nueva versión de las instrucciones cargar (LDX, LD IX) y almacenar (STX, LD(), IX) para colocar en el registro índice de la CPU un valor inicial de $00. El registro X del 6502 es un registro de un solo byte, pero el registro IX del Z80 es de dos bytes. Los registros índice poseen funciones especiales, pero esencialmente son RAM de la CPU, al igual que el acumulador, y aquí los utilizamos como acumuladores suplementarios donde colocaremos el contador de bucle. Cuando se produce la salida del bucle, el contenido del registro X del 6502 se almacenará en $5E20. En la versión Z80, el byte *lo* del registro (de dos bytes) IX se almacenará en $5E20 y el byte *hi* en $5E21.

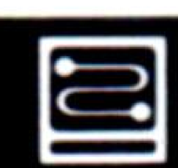
En tercer lugar, una instrucción completamente nueva ha ocupado el lugar de la instrucción ADC como comienzo (START) del bucle: tanto INX como INC IX son instrucciones de incremento, para que el contenido del registro índice aumente (o se INCremente) en razón de $01. Con ésta actualizamos el valor del contador del bucle cada vez que este último se ejecuta.

Podemos "traducir" así el programa: "poner el contador del bucle a cero, empezar el bucle incrementando el contador, agregar $34 al acumulador y bifurcar hacia atrás hasta el comienzo del bucle si el flag de arrastre se ha activado; de lo contrario almacenar el contenido del contador del bucle en $5E20". Una pequeña y ulterior modificación del programa aumentará enormemente su utilidad y su alcance:

6502		
0000	ORG	$5DFA
5DFA	LDX	#$00
5DFC START	STA	$5E22,X
5DFF	INX	
5E00	ADC	#$34
5E02	BCC	START
5E04	STX	$5E20
5E07 EXIT	RTS	

Z80		
0000	ORG	$5DF7
5DF7	LD	IX,$5E00
5DFB START	LD	(IX+$22),A
5DFE	INC	IX
5E00	ADC	A,$34
5E02	JR	NC,START
5E04	LD	($5E20),IX
5E08 EXIT	RET	

Liz Dixon



Saltos relativos
La mayoría de las instrucciones de bifurcación, como BCS ("bifurcar si el flag de arrastre está activado") y JR NZ ("saltar —*jump*— si el acumulador no es cero"), actúan en función de la situación del indicador de estado del procesador y utilizan la modalidad de salto relativo para reorientar el flujo de control a través del programa. La alternativa es el salto absoluto. En el ejemplo, la instrucción BCS $01 siempre produce un salto relativo de un byte hacia adelante (siempre, claro está, que se produzca efectivamente algún salto, pues depende del estado del flag de arrastre), con independencia de la dirección de posición en donde resida el lenguaje máquina. Aquí, la instrucción BCS $01 siempre va seguida por la instrucción INX, que en sí misma es una instrucción de byte único; por consiguiente, cuando el flag de arrastre está activado, se salta INX

Saltos absolutos
En este ejemplo, la instrucción JP $65A2 produce un salto incondicional cada vez que se pasa por ella. Su efecto consiste en dirigir de nuevo la ejecución del programa hasta la dirección que conforma su operando: $65A2, en este caso. No se indica ninguna condición y la dirección que tiene la posición de la instrucción en el momento de la ejecución carece de importancia; la ejecución del programa continúa siempre a partir de la dirección especificada.
Ambas modalidades de salto poseen sus ventajas y sus inconvenientes, pero el criterio más importante al elegir entre un salto relativo o un salto absoluto es la posibilidad de volverse a posicionar: en la programación en lenguaje assembly es bastante común escribir una rutina y ensamblarla en una dirección ORG, volviéndola a utilizar luego en la misma forma pero con un valor ORG distinto. Si todos los saltos de la rutina son relativos, entonces cambiar las direcciones de posición de las instrucciones no importará en absoluto y el programa fluirá uniformemente a lo largo de sus rutas previstas; no obstante, si alguno de los saltos es absoluto, cuando la rutina se ensamble en una ORG diferente, los saltos seguirán enviando el control a la dirección especificada, que bien podría no tener la más mínima importancia para la rutina. Los saltos relativos son reposicionables, los saltos absolutos no.

Tanto la versión 6502 como la Z80 producen el mismo efecto: crean en la posición $5E22 una tabla de almacenamiento de los sucesivos valores del acumulador a medida que se va ejecutando el programa, y finalmente almacenan en $5E20 el valor final del contador del bucle, que es, asimismo, el número de bytes de la tabla que comienza en $5E22.

La versión 6502 lo consigue con la instrucción STA $5E22,X, que significa "sumar el contenido del registro X a la dirección base, $5E22, luego almacenar el contenido del acumulador en la dirección así formada". Aquí la instrucción STA está en modalidad indexada directa absoluta (véase p. 677): es decir, se utiliza el registro X como un índice para modificar la dirección base, $5E22. Dado que el registro X está inicializado en $00 y se incrementa subsiguientemente a cada vuelta, el valor de comienzo del acumulador se almacenará en $5E22, el siguiente valor en $5E23, y así sucesivamente. Una vez decidida la salida del bucle, STX almacena el valor final del contador del bucle en la posición $5E20.

La versión Z80 utiliza el registro IX como un señalador de la dirección de almacenamiento corriente, mientras continúa utilizando al byte *lo* de IX como contador del bucle. La instrucción LD IX, $5E00 coloca la dirección base, $5E00, en el registro IX, de modo que el byte *lo* de IX contendrá $00. La instrucción LD (IX + $22),A, que tiene un aspecto tan peculiar, significa "sumar $22 a la dirección contenida en IX y almacenar el contenido del acumulador en la dirección así formada". Puesto que el registro IX está inicializado en $5E00, y se incrementa subsiguientemente a cada vuelta del bucle, el valor de comienzo del acumulador se almacenará en $5E22, el valor siguiente en $5E23, y así sucesivamente.

Mientras tanto, el byte *lo* de IX registra el número de vueltas del bucle, hasta que finalmente se almacena en $5E20 cuando termina el bucle. Aquí la instrucción LD (IX + $22),A está en la modalidad de direccionamiento indexada indirecta absoluta, que es bastante más complicada que la versión 6502 pero mucho más eficaz.

Hemos analizado con cierta profundidad el bucle y las estructuras de matriz en lenguaje assembly. Ambas técnicas de programación son de inmenso valor para el lenguaje máquina. En el próximo capítulo del curso las aplicaremos.

El próximo capítulo de nuestro curso de lenguaje máquina, en efecto, va a representar un nuevo e importante paso hacia adelante en este proceso cuya meta final será la generalización de nuestros programas en este código. Muchos fragmentos de programas pueden muy bien convertirse en subrutinas adaptables a las necesidades de la programación principal. No basta simplemente con saber cómo se realizan las bifurcaciones y los bucles, sino que debemos también conocer la manera en que las rutinas que hemos creado puedan "rescatarse" y a continuación utilizarse en cualquier momento que las necesitemos. Este planteamiento nos llevará al examen, en la siguiente lección del curso de lenguaje máquina, de uno de los dispositivos más útiles de la CPU, la pila.

Finalizada la investigación, el curso quedará completo en sus bases más imprescindibles prefijadas por nosotros, después de un estudio sencillo pero exhaustivo, por una parte, de las cuatro operaciones aritméticas por excelencia y, por otra, de las rutinas de visualización.

Las instrucciones de bifurcación condicionada, como hemos visto, dependen del contenido del registro de estado del procesador. Una de las razones por las que agregamos la opción de visualización binaria al programa monitor (véanse pp. 598 y 678) fue para permitirle a usted inspeccionar el contenido del PSR (indicador de estado) antes y después de ejecutar una instrucción, y observe los cambios en los flags. Ni en el 6502 ni en el Z80 hay ninguna instrucción para almacenar el contenido del PSR, de modo que hemos de utilizar estas órdenes:

estas órdenes:

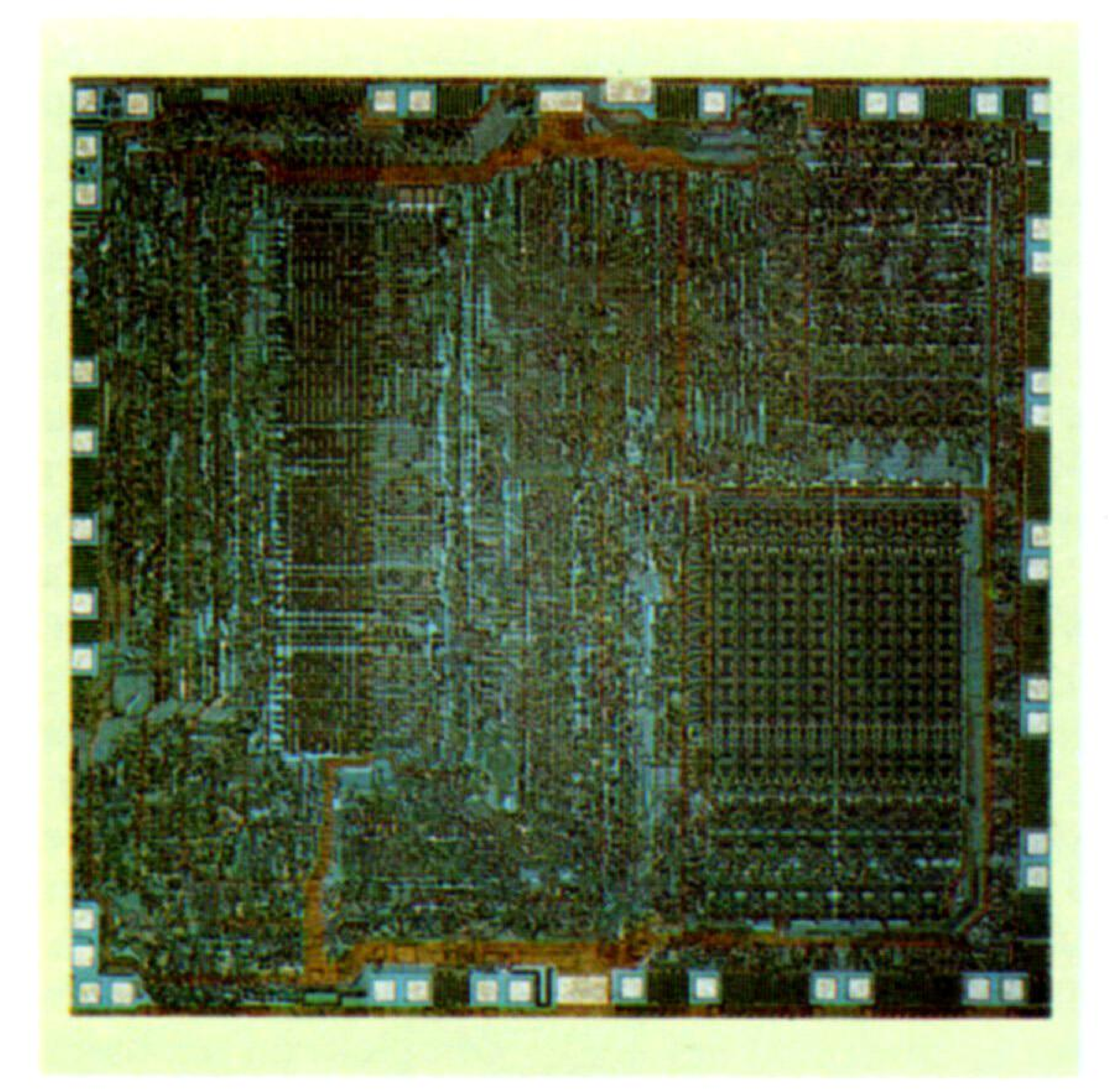
	Z80
3E00 F5	PUSH AF
3E01 F5	PUSH AF
3E02 E1	POP HL
3E03 22 lo hi	LD (STORE1),HL
3E06 F1	POP AF

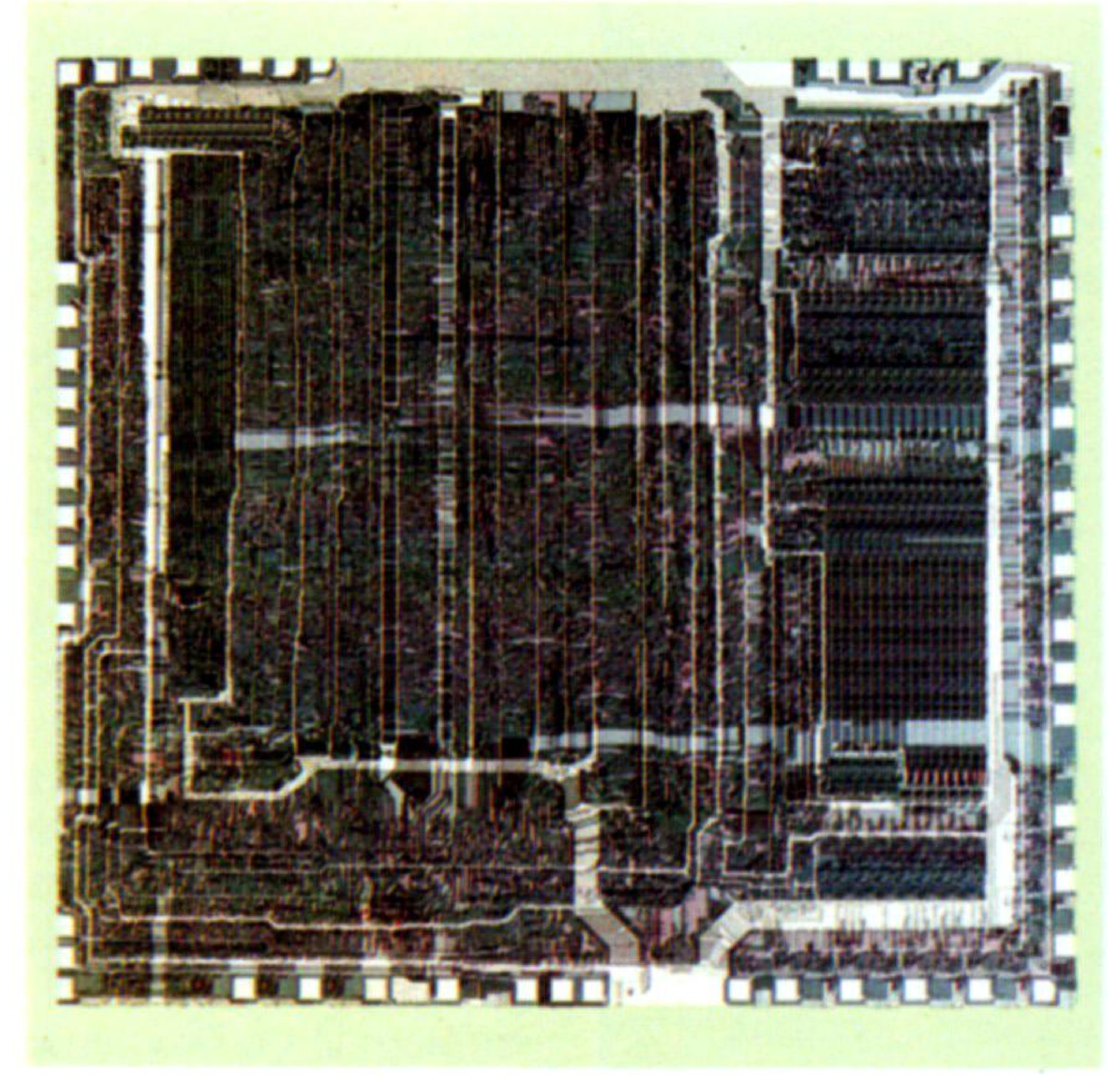
	6502
3E00 48	PHA
3E01 08	PHP
3E02 48	PHA
3E03 08	PHP
3E04 68	PLA
3E05 8D lo hi	STA STORE1
3E08 68	PLA
3E09 8D lo' hi'	STA (1+STORE1)
3E0C 28	PLP
3E0D 68	PLA

Esta secuencia de instrucciones hará que el contenido corriente del PSR se almacene en el byte direccionado por STORE1 (una dirección apropiada para su máquina), mientras que el contenido del acumulador se almacenará en (1 + STORE1). Para utilizar estas instrucciones, simplemente insértelas como un bloque antes y después de la instrucción del programa cuyo efecto usted desea observar. Debe acordarse, no obstante, de sumarle dos al valor de STORE1 cada vez que inserte este bloque. Después de haber ejecutado el programa puede emplear el monitor para visualizar la sección de la memoria en donde ha almacenado los diversos contenidos del PSR y del acumulador.
Quizá haya pensado que este bloque se debería tratar como una rutina en vez de darle entrada repetidamente donde se necesita. En lenguaje assembly hay un equivalente al GOSUB del BASIC, pero utilizarlo aquí complicaría las cosas, dado que emplea la pila y ésta interferiría con la utilización, por parte del bloque, de la misma lista (PLA, PUSH, PHP, etc., son todas manipulaciones de la pila, que explicaremos más adelante con detenimiento). Observe la diferencia, en cuanto a longitud, del código Z80 y el 6502: los responsables de esta variación son los registros de dos bytes del Z80 y las instrucciones relacionadas con los mismos.

En vanguardia

El microprocesador Z80, junto con su competidor, el 6502, están convirtiendo en realidad lo que se creía fantasía de la ciencia-ficción: un ordenador en cada hogar





Juego de chips
Estas fotomicrografías ilustran la complejidad de los circuitos de los microprocesadores. A la izquierda vemos un imagen enormemente ampliada del tan exitoso Z80 de Zilog y, a la derecha, un chip más reciente perteneciente a la serie Z8000. Observe que el Z8000, al ser un chip de 16 bits, tiene un diseño más denso y más conexiones alrededor del borde del chip

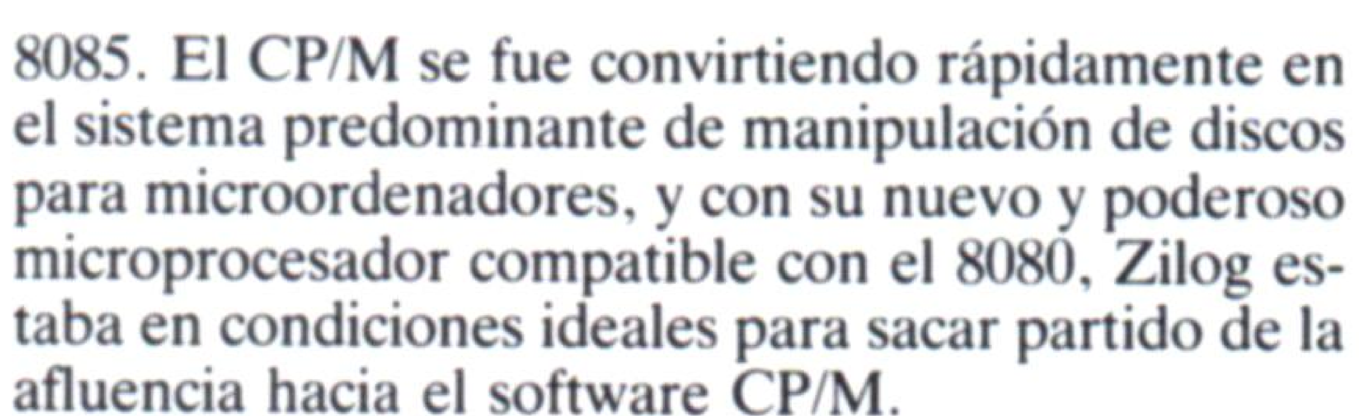


Frank de Weeger, presidente de Zilog

Cuando Zilog Incorporated introdujo el microprocesador Z-80 en 1977, casi nadie intuyó que comenzaba una revolución. Sin embargo, en pocos años, el Z-80 y su principal competidor, el 6502, convertirían en realidad lo que antes pareció mera fantasía: la presencia de un ordenador en cada hogar.

La historia de Zilog empieza a principios de los años setenta, cuando Frederico Faggin y Masatoshi Shima, dos empleados de la fábrica de microchips Intel, se independizan para formar su propia empresa. Ambos habían desempeñado un importante papel en el desarrollo del microchip 8080A (considerado como el primer "ordenador en un chip"), valiéndose de su experiencia empezaron a trabajar en un nuevo desarrollo del microprocesador. El 8080 ya se estaba haciendo muy popular entre los diseñadores y aficionados a la informática, de modo que Faggin y Shima, decidieron diseñar un nuevo chip que fuera compatible con el 8080. De esta manera, podría sacar partido de la gran cantidad de software que ya se había escrito para el 8080. Aprovechando el detallado conocimiento que poseían acerca del 8080, pudieron ampliar el juego de instrucciones (la lista de éstas en lenguaje máquina que contiene el microprocesador) mediante la instrucción de registros adicionales, opcodes de dos bytes y otras técnicas. El microprocesador, el Z80, representó una considerable mejora.

Esta revolución de hardware coincidió con una revolución paralela de software. En 1972, Gary Kildall y John Torode escribieron un programa llamado Control Program/Monitor, o CP/M, que permitió que un microprocesador manipulara unidades de disco flexible, introducidas recientemente. Dado que Kildall era consultor de Intel, el programa estaba diseñado para funcionar en el 8080 y el



8085. El CP/M se fue convirtiendo rápidamente en el sistema predominante de manipulación de discos para microordenadores, y con su nuevo y poderoso microprocesador compatible con el 8080, Zilog estaba en condiciones ideales para sacar partido de la afluencia hacia el software CP/M.

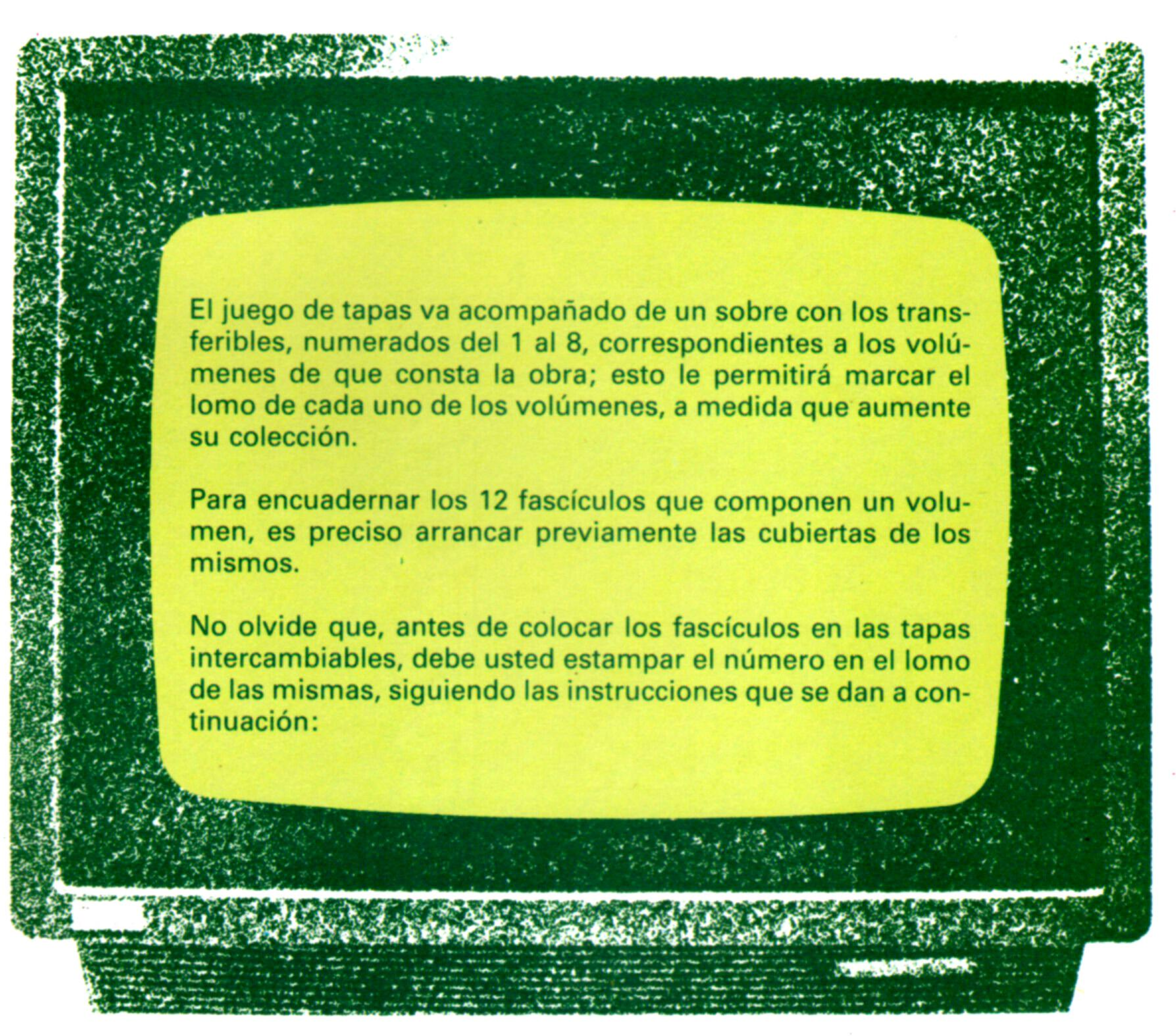
En los años siguientes el camino de Zilog no ha seguido la misma línea. A pesar de que el Z80 se sigue vendiendo en grandes cantidades (la empresa todavía produce alrededor de un millón por mes), los intentos por mejorar el chip para el mercado de 16 bits han tenido controvertidos resultados.

El primer intento de Zilog por conseguir un procesador de 16 bits fue el Z8000. Aunque por lo general se lo reconoce como un dispositivo muy potente, con un vasto juego de instrucciones y un gran número de registros, demostró ser un chip sumamente complejo de programar. El Z8000 se vio enfrentado a varios obstáculos importantes. Para empezar, a pesar de ser compatible con el Z80000, o Z80K (el procesador de 32 bits aún por lanzar al mercado), no lo era con el Z80 y, por lo tanto, no pudo sacar partido de la gama y la diversidad de programas que se habían escrito para éste en los años precedentes. Esto tuvo como consecuencia que aquellos fabricantes interesados en versiones mejoradas de 16 bits para sus máquinas se decidieran por incorporar microchips menos exigentes.

A la vista de la escasa aceptación del Z8000 en el mercado del microordenador, Zilog optó por diseñar un nuevo modelo. La empresa está a punto de lanzar el procesador de 16 bits Z800, que es compatible con el Z80. Las expectativas también parecen ser promisorias en otros aspectos: Commodore ha anunciado que su nueva gama de máquinas de oficina estará basada en el Z8000.



Con el próximo fascículo se pondrán a la venta las tapas correspondientes al tercer volumen.



El juego de tapas va acompañado de un sobre con los transferibles, numerados del 1 al 8, correspondientes a los volúmenes de que consta la obra; esto le permitirá marcar el lomo de cada uno de los volúmenes, a medida que aumente su colección.

Para encuadernar los 12 fascículos que componen un volumen, es preciso arrancar previamente las cubiertas de los mismos.

No olvide que, antes de colocar los fascículos en las tapas intercambiables, debe usted estampar el número en el lomo de las mismas, siguiendo las instrucciones que se dan a continuación:



1 Desprenda la hojita de protección y aplique el transferible en el lomo de la cubierta, haciendo coincidir los ángulos de referencia con los del recuadro del lomo.

2 Con un bolígrafo o un objeto de punta roma, repase varias veces el número, presionando como si quisiera borrarlo por completo.

3 Retire con cuidado y comprobará que el número ya está impreso en la cubierta. Cúbralo con la hojita de protección y repita la operación anterior con un objeto liso y redondeado, a fin de asegurar una perfecta y total adherencia.

Editorial Delta

Cada sobre de transferibles contiene una serie completa de números, del 1 al 8, para fijar a los lomos de los volúmenes. Ya que en cada volumen sólo aplicará el número correspondiente, puede utilizar los restantes para hacer una prueba preliminar.

Con el próximo número se ponen a la venta las tapas intercambiables para encuadernar 12 fascículos de

MI COMPUTER

Cada juego de tapas va acompañado de una colección de transferibles, para que usted mismo pueda colocar en cada lomo el número de tomo que corresponda



9 788485 822836
10035